

Deutsch



FUJITSU Software BS2000

# AID V3.4B

Basishandbuch

Benutzerhandbuch

Ausgabe Juni 2018

## **Kritik... Anregungen... Korrekturen...**

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

Sie können uns Ihre Kommentare per E-Mail an [manuals@ts.fujitsu.com](mailto:manuals@ts.fujitsu.com) senden.

## **Nach DIN EN ISO 9001:2015 zertifizierte Dokumentationserstellung**

Um eine gleichbleibend hohe Qualität und Anwenderfreundlichkeit zu gewährleisten, wurde diese Dokumentation nach den Vorgaben eines Qualitätsmanagementsystems erstellt, welches die Forderungen der DIN EN ISO 9001:2015 erfüllt.

cognitas. Gesellschaft für Technik-Dokumentation mbH  
[www.cognitas.de](http://www.cognitas.de)

## **Copyright und Handelsmarken**

Copyright © 2018 Fujitsu Technology Solutions GmbH.

Alle Rechte vorbehalten.

Liefermöglichkeiten und technische Änderungen vorbehalten.

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

---

# Inhalt

<b>1</b>	<b>Einleitung</b> . . . . .	<b>9</b>
<b>1.1</b>	<b>Zielsetzung und Zielgruppen der AID-Dokumentation</b> . . . . .	<b>9</b>
<b>1.2</b>	<b>Konzept der AID-Dokumentation</b> . . . . .	<b>10</b>
<b>1.3</b>	<b>Änderungen gegenüber dem Vorgänger-Handbuch</b> . . . . .	<b>11</b>
<b>1.4</b>	<b>Darstellungsmittel</b> . . . . .	<b>12</b>
<b>2</b>	<b>Metasyntax</b> . . . . .	<b>13</b>
<b>3</b>	<b>BS2000-Umgebung, Grundlagen und Kommandovorrat</b> . . . . .	<b>15</b>
<b>3.1</b>	<b>AID im BS2000</b> . . . . .	<b>15</b>
3.1.1	Laden von AID . . . . .	15
3.1.2	Anwenden von AID . . . . .	16
3.1.3	AID und der BS2000-Kommandointerpreter . . . . .	16
3.1.4	AID und SDF . . . . .	17
3.1.5	AID-Linknamen . . . . .	17
3.1.6	XS-Programmierung . . . . .	17
3.1.7	Programme im AR-Modus . . . . .	18
3.1.8	Test-Privilegierung . . . . .	19
<b>3.2</b>	<b>Grundlegende Begriffe</b> . . . . .	<b>20</b>
3.2.1	Das Testobjekt . . . . .	20
3.2.2	Objekt-Strukturliste und LSD . . . . .	20
3.2.3	Symbolisch - maschinennah . . . . .	21
3.2.4	Der AID-Arbeitsbereich . . . . .	21
3.2.5	Speicherobjekte und Speicherreferenzen . . . . .	22
3.2.6	Namenskonventionen bei AID . . . . .	23
3.2.7	Zeichendarstellung . . . . .	24
3.2.7.1	Zeichendarstellung durch UTF16 / UTFE . . . . .	24
3.2.7.2	Zeichendarstellung in durch XHCS unterstützten Coded Character Sets (CCS) . . . . .	24

<b>3.3</b>	<b>Die AID-Kommandos</b>	<b>25</b>
3.3.1	Ablaufüberwachung	27
3.3.2	Ablaufsteuerung	28
3.3.3	Ausgabe und Modifikation von Speicherinhalten	29
3.3.4	Verwaltungsfunktionen	30
3.3.5	Übersicht über die Geltungsdauer der Kommandos	32
<b>4</b>	<b>Voraussetzungen zum Testen mit AID</b>	<b>35</b>
<b>4.1</b>	<b>Testen auf Maschinencode-Ebene</b>	<b>36</b>
<b>4.2</b>	<b>Symbolisches Testen</b>	<b>37</b>
4.2.1	Übersetzen	39
4.2.2	Binden mit BINDER	39
4.2.3	Binden und Laden mit DBL	41
4.2.4	Nachladen von LSD-Sätzen durch AID	43
<b>5</b>	<b>Kommandoeingabe</b>	<b>45</b>
<b>5.1</b>	<b>Kommandoformat</b>	<b>45</b>
<b>5.2</b>	<b>Einzelkommandos</b>	<b>47</b>
<b>5.3</b>	<b>Kommandofolgen und Subkommandos</b>	<b>47</b>
<b>5.4</b>	<b>Kommandodateien</b>	<b>50</b>
<b>6</b>	<b>Subkommando</b>	<b>51</b>
<b>6.1</b>	<b>Beschreibung</b>	<b>51</b>
<b>6.2</b>	<b>Name und Durchlaufzähler</b>	<b>53</b>
<b>6.3</b>	<b>Bedingte Ausführung</b>	<b>55</b>
<b>6.4</b>	<b>Ketten</b>	<b>62</b>
<b>6.5</b>	<b>Schachteln</b>	<b>65</b>
<b>6.6</b>	<b>Löschen</b>	<b>67</b>

---

<b>7</b>	<b>Adressierung in AID</b>	<b>69</b>
<b>7.1</b>	<b>Qualifikationen</b>	<b>70</b>
7.1.1	Basisqualifikation	70
7.1.2	Bereichsqualifikationen	71
<b>7.2</b>	<b>Speicherreferenzen</b>	<b>78</b>
7.2.1	Maschinennahe Speicherreferenzen	80
7.2.2	Symbolische Speicherreferenzen	82
7.2.2.1	Datennamen	82
7.2.2.2	Anweisungsnamen und Source-Referenzen	85
7.2.3	Schlüsselwörter	87
7.2.4	Komplexe Speicherreferenzen	88
7.2.4.1	Adressversatz "."	89
7.2.4.2	Indirekte Adressierung "->" / "***"	91
7.2.4.3	Typmodifikation	95
7.2.4.4	Längenmodifikation	98
7.2.4.5	Arithmetischer Ausdruck	100
7.2.4.6	Adress-, Typ- und Längenselektor	102
7.2.4.7	Besonderheiten beim Zusammenwirken der verschiedenen Bestandteile	104
<b>8</b>	<b>Operand Medium-und-Menge</b>	<b>107</b>
<b>9</b>	<b>AID-Literale</b>	<b>113</b>
<b>9.1</b>	<b>Zeichen-Literale</b>	<b>113</b>
9.1.1	Character-Literal	113
9.1.1.1	Eingabeformate	113
9.1.1.2	Zeichencodierung	114
9.1.1.3	Konvertierungsfunktion %C() und %UTF16()	114
9.1.1.4	Suchen nach Character-Literalen mit %FIND	115
9.1.2	Sedezimal-Literal	119
9.1.3	Binär-Literal	120
<b>9.2</b>	<b>Numerische Literale</b>	<b>121</b>
9.2.1	Ganzzahl	121
9.2.2	Sedezimalzahl	121
9.2.3	Dezimalpunktzahl	122
9.2.4	Gleitpunktzahl	123

<b>10</b>	<b>Schlüsselwörter</b> . . . . .	<b>125</b>
<b>10.1</b>	<b>Allgemeine Speichertypen</b> . . . . .	<b>125</b>
10.1.1	Speichertypen zur Invertierung der Byte-Reihenfolge eines Datenelements . . . . .	126
<b>10.2</b>	<b>Speichertypen zur Interpretation von Maschinenbefehlen</b> . . . . .	<b>128</b>
<b>10.3</b>	<b>Programmregister und Befehlszähler</b> . . . . .	<b>129</b>
<b>10.4</b>	<b>AID-Register</b> . . . . .	<b>130</b>
<b>10.5</b>	<b>Speicherklassen</b> . . . . .	<b>130</b>
<b>10.6</b>	<b>Systeminformationen</b> . . . . .	<b>130</b>
<b>10.7</b>	<b>Durchlaufzähler</b> . . . . .	<b>133</b>
<b>10.8</b>	<b>Logische Werte</b> . . . . .	<b>133</b>
<b>10.9</b>	<b>Vorschubsteuerung</b> . . . . .	<b>133</b>
<b>10.10</b>	<b>Adressumschaltung</b> . . . . .	<b>134</b>
<b>10.11</b>	<b>Aktuelle Aufrufhierarchie</b> . . . . .	<b>134</b>
<b>10.12</b>	<b>Kriterium bei %CONTROLn und %TRACE</b> . . . . .	<b>134</b>
<b>10.13</b>	<b>Ereignis bei %ON</b> . . . . .	<b>135</b>
<b>11</b>	<b>Spezielle Anwendungen</b> . . . . .	<b>139</b>
<b>11.1</b>	<b>%ON und STXIT</b> . . . . .	<b>139</b>
<b>11.2</b>	<b>Programme mit Überlagerungsstruktur</b> . . . . .	<b>140</b>
<b>12</b>	<b>Einschränkungen und Wechselwirkungen</b> . . . . .	<b>141</b>
<b>12.1</b>	<b>%ON %WRITE mit %INSERT, %CONTROLn und %TRACE</b> . . . . .	<b>141</b>
<b>12.2</b>	<b>Wechselwirkungen zwischen Ablaufüberwachung und Ausgabe oder Modifikation von Speicherinhalten</b> . . . . .	<b>142</b>
<b>12.3</b>	<b>Testpunkte in Common Memory Pools</b> . . . . .	<b>143</b>
<b>12.4</b>	<b>Low-Level-Trace und -Control in Verbindung mit Contingencies</b> . . . . .	<b>144</b>
12.4.1	%TRACE . . . . .	144
12.4.2	%CONTROL . . . . .	144

<b>13</b>	<b>Anhang</b> . . . . .	<b>145</b>
<b>13.1</b>	<b>In Kommandofolgen und Subkommandos unzulässige SDF- und ISP-Kommandos</b> . . . . .	<b>145</b>
<b>13.2</b>	<b>Ereigniscodes</b> . . . . .	<b>148</b>
	<b>Fachwörter</b> . . . . .	<b>149</b>
	<b>Literatur</b> . . . . .	<b>159</b>
	<b>Stichwörter</b> . . . . .	<b>161</b>





---

# 1 Einleitung

Mit AID (Advanced Interactive Debugger) steht im Betriebssystem BS2000 eine leistungsstarke Dialog-Testhilfe zur Verfügung. Fehlerdiagnose, Test und vorläufige Fehlerbehebung aller im BS2000 erstellten Programme können Sie mit AID wesentlich schneller und mit weniger Aufwand durchführen als mit anderen Mitteln, wie z.B. dem Einfügen von Testhilfe-Anweisungen im Programm. AID ist permanent verfügbar und besitzt eine hohe Anpassungsfähigkeit an die jeweilige Programmiersprache. Ein Programm, das Sie mit AID getestet haben, muss nicht immer 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. Außerdem können Sie Informationen über den Programmablauf und zur Handhabung von AID abfragen.

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

## 1.1 Zielsetzung und Zielgruppen der AID-Dokumentation

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

## 1.2 Konzept der AID-Dokumentation

Die Dokumentation von AID besteht aus einem Basishandbuch und den sprachspezifischen Handbüchern für das symbolische Testen sowie dem Handbuch für das Testen auf Maschinencode-Ebene. Zusätzlich liegt für den geübten AID-Anwender als Nachschlagewerk ein [Tabellenheft \[7\]](#) vor, in dem die Syntax aller Kommandos und die Operanden mit kurzen Erläuterungen aufgeführt sind. Außerdem zeigt das Tabellenheft die %SET-Tabellen. Zusammen mit dem Basishandbuch enthält das Handbuch für die von Ihnen gewählte Sprache alle Informationen, die Sie zum Testen brauchen. Das Handbuch für das Testen auf Maschinencode-Ebene kann statt oder zusätzlich zu einem der sprachspezifischen Handbücher eingesetzt werden.

### **AID - Basishandbuch**

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

### **AID-Benutzerhandbücher**

In den Benutzerhandbüchern finden Sie die Kommandos in alphabetischer Reihenfolge. Alle einfachen Speicherreferenzen sind in diesen Handbüchern beschrieben.

[AID - Testen von COBOL-Programmen \[2\]](#)

[AID - Testen von FORTRAN-Programmen \[3\]](#)

[AID - Testen unter Posix \[5\]](#)

[AID - Testen von ASSEMBH-Programmen \[5\]](#)

[AID - Testen von C/C++-Programmen \[6\]](#)

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

Die zusätzlichen Möglichkeiten des „maschinennahen Testens“ sind beschrieben in [„AID - Testen auf Maschinencode-Ebene“ \[1\]](#).

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

## Readme-Datei

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

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

### *Informationen unter BS2000*

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

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

Diese Datei enthält eine kurze Information zur Readme-Datei in deutscher oder englischer Sprache (<lang>=D/E). Die Information können Sie am Bildschirm mit dem Kommando `SHOW-FILE` oder mit einem Editor ansehen.

Das Kommando `/SHOW-INSTALLATION-PATH INSTALLATION-UNIT=<product>` zeigt, unter welcher Benutzerkennung die Dateien des Produkts abgelegt sind.

### *Ergänzende Produkt-Informationen*

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

## 1.3 Änderungen gegenüber dem Vorgänger-Handbuch

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

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

## 1.4 Darstellungsmittel

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



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

---

## 2 Metasyntax

Für die Darstellung von Kommandos und deren Operanden 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.

$$\left\{ \begin{array}{l} \text{alternativ} \\ \dots \\ \text{alternativ} \end{array} \right\}$$

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

---

## 3 BS2000-Umgebung, Grundlagen und Kommandovorrat

### 3.1 AID im BS2000

Das Testsystem AID besteht aus zwei Komponenten

- der Anwenderschnittstelle AID und
- der Systemschnittstelle AIDSYS.

Durch diese Aufteilung ist die Anwenderschnittstelle AID unabhängig von den BS2000-Versionen. Alle erforderlichen Systemfunktionen werden über AIDSYS abgewickelt. Diese Unabhängigkeit von BS2000-Versionen ist von Bedeutung, wenn Sie sich z.B. aus einem Dump, der auf einem anderen Rechner mit einer anderen BS2000-Version erzeugt wurde, eine der möglichen Systemtabellen aufbereitet ausgeben lassen. Die versionsabhängige Aufbereitung erfolgt über AIDSYS, das in der Lage ist festzustellen, mit welcher BS2000-Version der Speicherabzug erzeugt wurde, und dann die angeforderte Systemtabelle entsprechend aufbereitet an AID zur Ausgabe übergibt.

Die Eingabe von AID-Kommandos und die Ausgabe von AID-Meldungen erfolgt über die Systemdateien SYSCMD und SYSOUT wie bei BS2000-Kommandos (siehe [Kommandos \[8\]](#)).

#### 3.1.1 Laden von AID

AID wird nicht durch den einzelnen Anwender, sondern durch die Systembetreuung mit dem Kommando /START-SUBSYSTEM AID (siehe Handbuch „[Kommandos \[8\]](#)“) geladen. Danach steht AID jedem Anwender zur Verfügung, ohne dass er weitere Vorkehrungen treffen muss.

### 3.1.2 Anwenden von AID

Im folgenden sind zwei Arten beschrieben, wie eine Testsitzung mit AID beginnen kann:

1. Sie laden und starten Ihr Programm. Wenn der Programmablauf durch einen Fehler unterbrochen wird, laden Sie, um symbolisch testen zu können, die LSD-Sätze mit dem %SYMLIB-Kommando zu der Übersetzungseinheit, in der der Fehler aufgetreten ist. Nun können Sie AID-Kommandos eingeben.

```
/START-EXECUTABLE-PROGRAM FROM-FILE=...
...
% IDA0N51 PROGRAM INTERRUPT AT LOCATION '000B62 (MOBS), (CDUMP), EC=58'
% IDA0N45 DUMP DESIRED? REPLY (Y = USER/AREA DUMP; Y,SYSTEM = SYSTEM DUMP;
N = NO)
...
/%SYMLIB ...
/%SDUMP %NEST
...
```

2. Sie laden Ihr Programm. Wenn Sie symbolisch testen wollen, geben Sie den Parameter an, der die LSD-Sätze gleich mitlädt. Sie geben AID-Kommandos zur Überwachung ein und starten das Programm mit einem AID-Kommando.

```
/LOAD-EXECUTABLE-PROGRAM FROM-FILE=..., TEST-OPTIONS=AID
/%INSERT ...
/%R
...
```

### 3.1.3 AID und der BS2000-Kommandointerpreter

Die AID-Funktionen rufen Sie mit AID-Kommandos auf. Ein AID-Kommando beginnt mit einem %-Zeichen, unmittelbar gefolgt vom Kommandonamen:

```
%DISPLAY ARRAY1
```

Immer, wenn sich Ihre Task im Kommandomodus befindet, können Sie auch AID-Kommandos eingeben. Die AID-Kommandos werden wie die BS2000-Kommandos vom Kommandointerpreter des BS2000 übernommen. Der identifiziert sie anhand des %-Zeichens als AID-Kommandos und übergibt sie dann an AID.

AID-Kommandos können Sie im Dialog oder in Prozedurdateien eingeben. Über den Makro CMD können Sie auch aus dem Programm AID-Kommandos aufrufen (siehe „[Makroaufrufe an den Ablaufteil](#)“ [10]).

In Kommandofolgen können Sie sowohl AID-Kommandos als auch BS2000-Kommandos verwenden.



### 3.1.4 AID und SDF

SDF (System Dialog Facility) ist die Dialogschnittstelle zum BS2000. Zu SDF gehört eine eigene Kommandosprache. Vor SDF wurde die Kommandosprache ISP (Interactive String Processor) verwendet. Im AID-Basishandbuch und in den sprachspezifischen Handbüchern werden BS2000-Kommandos generell im SDF-Format und zwar in der EXPERT-Form angegeben (siehe „[Dialogschnittstelle SDF](#)“ [14]). In einigen Fällen finden Sie eine Gegenüberstellung mit den entsprechenden ISP-Kommandos im Anhang. Darauf wird dann jeweils hingewiesen.

Für AID-Kommandos gibt es keine SDF-Notation.

### 3.1.5 AID-Linknamen

Dump-Dateien können mit AID über die Linknamen D0 bis D7 angesprochen werden.

Datenausgaben, TRACE-Protokolle und REPs können Sie in Ausgabedateien schreiben lassen. AID-Ausgabedateien haben das Format `FCBTYPE=SAM, RECFORM=V, OPEN=EXTEND`. AID-Ausgabedateien weisen Sie über die Linknamen F0 bis F7 zu.

### 3.1.6 XS-Programmierung

In allen BS2000-Versionen können Programme den erweiterten Adressraum von über 16 Megabyte bis zu 2 Gigabyte nutzen. Entsprechend können Sie mit AID auch Programme im erweiterten Adressraum testen.

AID stellt sich automatisch auf den Adressierungsmodus des Testobjekts ein und kann sowohl mit 24-Bit-Adressen (unterer Adressraum) als auch mit 31-Bit-Adressen (erweiterter Adressraum) arbeiten.

Für den Fall, dass Sie z.B. Programmverknüpfungen von Modulen mit unterschiedlichem A-Modus testen, bietet AID folgende Funktionen an (siehe „[AID - Testen auf Maschinencode-Ebene](#)“ [1]):

- Schlüsselwort für die Systeminformation AMODE (%AMODE)
- Anzeigen des aktuellen A-Modus (%DISPLAY)
- Umschalten des A-Modus für das Testobjekt (%MOVE)
- Umschalten der Adressinterpretation bei indirekter Adressierung (%AINT)

### 3.1.7 Programme im AR-Modus

Anwenderprogramme können neben dem Programmraum (program space) weitere Adressräume für Daten, die Datenräume (data spaces) nutzen. Datenräume können nur Daten enthalten, Programmcode kann in einem Datenraum nicht ausgeführt werden. Ihre eindeutige Ansprache ist über die SPID (space identification) oder über einen bzw. mehrere ALETs (access list entry token) möglich. Für die Adressierung mit ALETs wurden die Zugriffsregister als weiterer Registersatz parallel zu den Mehrzweckregistern eingeführt. In den Zugriffsregistern sind die ALETs enthalten. Wenn der AR-Modus (access register mode) eingeschaltet ist, werden bei der Adressumsetzung in einem Maschinenbefehl die Zugriffsregister mit ausgewertet und so Daten in einem Datenraum adressiert. Details siehe Handbuch „[Makroaufrufe an den Ablaufteil](#)“ [10].

Für Programme im AR-Modus bietet AID folgende Funktionen an (siehe Handbuch „[AID - Testen auf Maschinencode-Ebene](#)“ [1]):

- Schlüsselwort für die Systeminformation ASC-Modus (%ASC) zur Abfrage des AR-Modus.
- Schlüsselwörter für die Zugriffsregister (%nAR, %AR)
- ALET- und SPID-Qualifikation zur eindeutigen Ansprache von virtuellen Adressen in Datenräumen
- Schlüsselwort für die Systeminformation über die aktiven Datenräume (%DS[(ALET/SPID-qua)])
- Kennzeichnung von virtuellen Adressen aus Datenräumen mit einem Stern "\*" bei der Ausgabe mit %DISPLAY und im %TRACE-Protokoll.

Daten in Datenräumen können nur über ihre virtuellen Adressen angesprochen werden. Wenn die Daten symbolisch aufbereitet werden sollen, können Sie dies über eine abschließende Typmodifikation erreichen.

### 3.1.8 Test-Privilegierung

Es ist notwendig, sicherzustellen, dass ein AID-Anwender nicht auf beliebige Datenbestände und Speicherbereiche innerhalb des Systems zugreifen oder sie sogar verändern kann. Für die Steuerung der Zugriffsrechte gibt es in jedem Benutzereintrag die Angabe „Test-Privilegierung“ für die Lese- und Schreib-Zugriffsberechtigung. Diese wird vom Systemverwalter eingetragen.

Beim Start einer Task werden immer die niedrigsten Privilegierungen (1,1) vergeben. Damit können Sie den gesamten Leistungsumfang von AID nutzen, wie er in der Dokumentation beschrieben ist.

Wenn Sie Dateien oder Bibliotheken mit READ- oder EXEC-PASSWORD geschützt haben, können Sie auch mit AID nur zugreifen, wenn Sie das Kennwort eingegeben haben.

Wenn Sie auf Speicherbereiche zugreifen wollen, die eine höhere Privilegierung benötigen, können Sie mit dem Kommando /MODIFY-TEST-OPTIONS Ihre Privilegierung verändern, sofern es der Eintrag im Benutzerkatalog erlaubt. Diesen Eintrag können Sie sich mit dem Kommando SHOW-USER-ATTRIBUTES anschauen (siehe „[Kommandos](#)“ [8]).

Außerdem bietet Ihnen AID Schlüsselwörter an, mit denen Sie auf geschützte Bereiche, z.B. Register, zugreifen können.

## 3.2 Grundlegende Begriffe

### 3.2.1 Das Testobjekt

Das Programm, das Sie mit AID bearbeiten wollen, wird als Testobjekt bezeichnet. Es kann unter Ihrer Benutzerkennung geladen sein oder als Speicherabzug in einer Dump-Datei vorliegen. Innerhalb einer Testsitzung können Sie zwischen diesen Möglichkeiten wechseln, um z.B. Daten im geladenen Programm mit Daten in einer Dump-Datei oder um Speicherabzüge aus unterschiedlichen Versionen desselben Objekts zu vergleichen.

Das Programm können Sie immer auf Maschinencode-Ebene testen und, wenn bei der Übersetzung LSD-Sätze erzeugt wurden, auch auf symbolischer Ebene. Hierzu brauchen Sie das Programm nicht erneut zu übersetzen oder zu binden. Da Sie das Programm ohne die symbolischen Informationen laden können, sind nach einem fehlerfreien Testablauf keine weiteren Übersetzungs- oder Binderläufe nötig. Das Programm können Sie sofort in den produktiven Einsatz übernehmen.

### 3.2.2 Objekt-Strukturliste und LSD

AID arbeitet mit zwei Listen, aus denen es Informationen über das Programm entnimmt:

Die Objekt-Strukturliste wird beim Binden mit BINDER standardmäßig aus dem ESV (External Symbols Vector) erzeugt. In der Objekt-Strukturliste sind unter anderem Informationen über die CSECTs, DSECTs und COMMONs eines Programms enthalten. Ist diese Liste vorhanden, können Sie über den Namen einer CSECT oder eines COMMONs auf die zugehörige Adresse, den Inhalt und die Länge zugreifen.

Die LSD (List for Symbolic Debugging) ist das Verzeichnis der im Modul definierten Datennamen, Anweisungsnamen und Namen von Programmteilen. Ebenso enthält sie die vom Compiler erzeugten Source-Referenzen. LSD-Sätze werden vom Compiler erzeugt, wenn Sie die entsprechende Option beim Übersetzen angeben. Wurden LSD-Sätze erzeugt, können Sie über die im Quellprogramm definierten und die vom Compiler erzeugten Namen auf Adresse, Inhalt, Länge und Typ der entsprechenden Speicherobjekte zugreifen.

Der Klasse-5-Speicherbedarf eines Programms mit LSD-Sätzen steigt je nach der Menge der symbolischen Definitionen auf ein Vielfaches der reinen Programmgröße. Speichern Sie die Bindemodule in einer PLAM-Bibliothek ab, dann können Sie das Programm ohne LSD-Sätze laden und starten und die PLAM-Bibliothek, die die LSD-Sätze enthält, mit dem Kommando %SYMLIB öffnen. Bei Bedarf lädt AID dann die LSD-Sätze aus der zugewiesenen Bibliothek nach.

### 3.2.3 Symbolisch - maschinennah

AID kennt zwei Testebenen: Auf symbolischer Ebene werden die vom Compiler erzeugten symbolischen Adressen aus den LSD-Sätzen benutzt. Speicherstellen sprechen Sie über die Namen an, die im Quellprogramm vergeben wurden. AID-Ausgaben enthalten die Programm-, Daten-, Anweisungsnamen und Source-Referenzen. Das Schlüsselwort %HLLOC (High-Level-Location) als Operand des AID-Kommandos %DISPLAY gibt die symbolischen Lokalisierungsinformationen aus. Sie bestehen aus dem Kontextnamen, dem Namen der Übersetzungseinheit und des aktuellen Haupt- oder Unterprogramms sowie der Source-Referenz, der die Adresse zugeordnet ist. Die AID-Kommandos %JUMP, %SDUMP und %SYMLIB können Sie nur auf symbolischer Ebene einsetzen, also wenn LSD-Sätze zu dem angesprochenen Programmteil vorhanden sind. Bei den Kommandos %CONTROL<sub>n</sub> und %TRACE entscheidet das Schlüsselwort für *kriterium*, ob die Überwachung oder Verfolgung auf symbolischer Ebene stattfindet.

Wenn Sie eine Adresse von AID errechnen lassen (komplexe Speicherreferenz), können Sie jederzeit von der symbolischen zur maschinennahen Ebene wechseln. Mit Adressselektion und Pointer-Operator ( %@(name)-> ) können Sie das angesprochene Speicherobjekt mit allen seinen maschinennahen Eigenschaften benutzen.

Auf Maschinencode-Ebene werden nur die CSECT- und COMMON-Informationen aus der Objekt-Strukturliste benutzt. AID-Ausgaben enthalten virtuelle Adressen und CSECT- bzw. COMMON-Namen. Das Schlüsselwort %LOC (Low-Level-Location) als Operand des AID-Kommandos %DISPLAY gibt die maschinennahen Lokalisierungsinformationen aus. Sie bestehen aus dem Kontextnamen, den Namen der Ladeeinheit, des Objektmoduls, der CSECT bzw. des COMMON sowie aus der CSECT- bzw. COMMON-relativen Adresse. Bei den Kommandos %CONTROL<sub>n</sub> und %TRACE entscheidet das Schlüsselwort für *kriterium*, ob die Überwachung oder Verfolgung auf Maschinencode-Ebene stattfindet.

In einer komplexen Speicherreferenz können Sie symbolische Adressen und Elemente der maschinennahen Adressierung verbinden und dabei alle Eigenschaften der symbolischen Adressierung weiterhin benutzen.

### 3.2.4 Der AID-Arbeitsbereich

Der AID-Arbeitsbereich ist der Adressraum, in dem Sie Speicherobjekte ohne Angabe einer Basisqualifikation ansprechen können. Er umfasst den nicht-privilegierten Teil des virtuellen Speichers in Ihrer Task, der vom Programm zusammen mit seinen angeschlossenen Subsystemen belegt ist oder den entsprechenden Bereich in einem Speicherabzug. Ob Sie im geladenen Programm oder in einem Speicherabzug testen, können Sie durch das Kommando %BASE festlegen. Ohne Angabe von %BASE liegt der AID-Arbeitsbereich im geladenen Programm. Dies wird als AID-Standard-Arbeitsbereich bezeichnet. Sie können auch innerhalb eines Kommandos vom aktuell eingestellten Arbeitsbereich abweichen, indem Sie in einem Adressoperanden eine entsprechende Basisqualifikation {E=VM|Dn} angeben.

Liegt der AID-Arbeitsbereich in einer Dump-Datei, können Sie die Kommandos zur Ablaufüberwachung und zur Ablaufsteuerung nicht einsetzen. Außerdem können Sie eine Dump-Datei nicht mit AID-Kommandos modifizieren. Sie können sich jedoch aus einer Dump-Datei Daten ausgeben lassen, die Aufrufhierarchie zum Zeitpunkt der Programmunterbrechung zurückverfolgen, Maschinencode in symbolische Assembler-Notation rückübersetzen, und Sie können Zeichenfolgen in einer Dump-Datei suchen. Außerdem können Speicherinhalte eines geladenen Programms mit Daten aus einer Dump-Datei überschrieben werden.

### 3.2.5 Speicherobjekte und Speicherreferenzen

Als Speicherobjekt wird eine bestimmte Anzahl von Bytes bezeichnet, die zusammenhängend ab einer bestimmten Adresse im Speicherbereich des Programms liegen. Das sind zum einen die Daten eines Programms und zum anderen der Befehlscode. Die außerhalb des Programmspeichers liegenden Register und der Befehlszähler gehören ebenfalls zu den Speicherobjekten. Sie werden von AID über Schlüsselwörter angesprochen.

Konstanten werden nicht als Speicherobjekte bezeichnet. 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.

Mit einer Speicherreferenz sprechen Sie ein Speicherobjekt an. Es gibt einfache Speicherreferenzen und komplexe Speicherreferenzen. Einfache Speicherreferenzen sind virtuelle Adressen, Namen, zu denen sich AID die Adresse aus den LSD-Sätzen holen kann, und Schlüsselwörter.

In einer komplexen Speicherreferenz errechnet AID eine Adresse gemäß Ihren Angaben, denen AID gleichzeitig Typ und Länge des durch diese Adresse bezeichneten Speicherobjekts entnimmt. Folgende Operationen können in einer komplexen Speicherreferenz vorkommen: Adressversatz, indirekte Adressierung, Typmodifikation, Längenmodifikation und Adressselektion.

Liegt eine Speicherreferenz nicht im gerade gültigen AID-Arbeitsbereichs oder außerhalb des aktuellen Haupt- oder Unterprogramms oder ist sie da drin nicht eindeutig, können Sie mit Qualifikationen den Pfad zu der gewünschten Speicherreferenz beschreiben.

### 3.2.6 Namenskonventionen bei AID

Für alle Namen, die Sie in AID-Kommandos einsetzen, um Programme oder Programmteile, Daten oder Anweisungen anzusprechen oder um Subkommandos zu benennen, gilt unabhängig von der verwendeten Programmiersprache der folgende Zeichenvorrat: a-z, A-Z, 0-9, \$, #, @, Unterstrich "\_" oder Bindestrich "-".

Der Bindestrich ist nicht als erstes Zeichen zugelassen und ist auch nur dann als Teil des Namens erlaubt ist, wenn SYMCHARS=STD eingestellt ist (Kommando %AID). Wenn der Bindestrich das letzte Zeichen eines Namens ist, können Sie diesen Namen nur mit N'...' angeben.

Generell müssen alle Namen, die Sonderzeichen enthalten oder für AID zweideutig sein können, in N'...' gesetzt werden. Marken mit Sonderzeichen sowie Marken als Ausgangsadresse für eine komplexe Speicherreferenz schreiben Sie in L'...'.

Um AID zu veranlassen, zwischen Groß-/Kleinschreibung zu unterscheiden, müssen Sie zunächst das Kommando %AID LOW[=ON] eingeben.

Bei BLS-Namen, also den Namen, die dem Binder-Lader-Starter bekannt sind wie z.B. Namen von CSECTs, COMMONs oder Entrys sowie bei den Namen von Übersetzungseinheiten (bei Fortran: Programmeinheiten) wird die Groß-/Kleinschreibung jedoch nur dann berücksichtigt, wenn das Kommando %AID LOW=ALL eingegeben wurde.

Erlaubte Länge ist für BLS-Namen und Namen von Übersetzungseinheiten 32 Zeichen, Namen von Daten und Programmteilen wie Funktionen, Prozeduren oder Unterprogrammen dürfen bis zu 255 Zeichen lang sein. Namen von Subkommandos können einschließlich der vorangestellten Zeichen "%•" bis zu 32 Zeichen lang sein.

#### Übersicht

Namen	Länge (max.)	%AID LOW=ON wirksam	%AID LOW=ALL wirksam
BLS-Namen und Namen von Übersetzungseinheiten	32	nein	ja
Daten- und Programmnamen	255	ja	ja
Namen von Marken	255	ja	ja
Subkommandonamen	32 einschl. %•	nein	nein

### 3.2.7 Zeichendarstellung

AID unterstützt die Unicode-Zeichendarstellung durch UTF16 / UTFE sowie die Zeicheninterpretation in Zeichensätzen, die durch XHCS unterstützt werden.

#### 3.2.7.1 Zeichendarstellung durch UTF16 / UTFE

Mit Unterstützung von Unicode gibt es in AID den Datentyp %UTF16 zur Darstellung von Strings. Bei diesem Datentyp hat jedes Zeichen eine 2-Byte-Verschlüsselung. Der bisher von AID unterstützte Datentyp zur Darstellung von Strings hat eine 1-Byte-EBCDIC-Codierung.

AID unterstützt für Eingabe- und Ausgabemedien die UTFE-Zeichencodierung. Diese Codierung ist die EBCDIC-Variante von UTF8, die eine Mehrbyte-Codierung in variabler Bytelänge unterstützt.

#### Einstellung einer EBCDIC-Codiertabelle über %AID

Mit dem Operanden EBCDIC des Kommandos %AID kann die EBCDIC-Codierung eines C-Strings spezifiziert werden, die AID verwendet, wenn eine Konvertierung zwischen einem UTFE-/%UTF16-String und einem 1-Byte-C-String durchgeführt werden muss.

AID unterstützt alle 1-Byte-EBCDIC-Codierungen, die das Subsystem XHCS-SYS anbietet. Die aktuellen Namen der Codiertabellen können Sie sich mit dem Kommando %SHOW %CCSN ausgeben lassen.

Mit den neuen Funktionen %C(...) und %UTF16(...) können Sie z.B. die Literal-Codierung in eine andere Codierung umwandeln.

#### 3.2.7.2 Zeichendarstellung in durch XHCS unterstützten Coded Character Sets (CCS)

Mit dem Operanden *CCS* des Kommandos %AID stellen Sie einen Zeichensatz, der durch XHCS unterstützt wird, ein, in dem die Zeichen interpretiert werden, falls keine explizite Angabe im %DISPLAY Kommando gemacht wird. Unicode-Zeichensätze sind nicht erlaubt.



### 3.3 Die AID-Kommandos

AID verfügt über eine Vielzahl von Funktionen, die Sie über AID-Kommandos aufrufen. Dieses Kapitel gibt Ihnen einen Überblick über den Leistungsumfang von AID. Die vollständigen Beschreibungen der Kommandos finden Sie in den sprachspezifischen Handbüchern oder dem Handbuch für das Testen auf Maschinencode-Ebene. Den AID-Kommandovorrat kann man in vier Funktionsgruppen einteilen. In der folgenden Übersicht sind alle AID-Kommandos mit ihren Operanden aufgeführt. Kommandos, die nur für das Testen auf symbolischer Ebene verwendet werden können, sind in der zweiten Spalte mit 'SY' gekennzeichnet.

#### Ablaufüberwachung

Kommandoname		Operanden
%C[ONTR]OL] <i>n</i>		[kriterium][,...] [IN control-bereich] <subkdo>
%I[N]SERT]		testpunkt [<subkdo> ] [steuerung]
%O[N]		{write-ereignis   ereignis} [<subkdo>]
%R[EM]O[V]E]		ziel

#### Ablaufsteuerung und Ablaufprotokollierung

Kommandoname		Operanden
%C[ON]T[IN]UE]		
%J[UM]P	SY	fortsetzung
%R[ES]UME]		
%S[T]OP		
%T[RACE]		[anzahl] [kriterium][,...] [IN trace-bereich]

**Ausgabe und Modifikation von Speicherinhalten**

Kommandoname		Operanden
%D[IS]A[SSEMBLE]		[ausgabe-menge] [FROM start]
%D[ISPLAY]		daten {,...} [medium-u-menge][,...]
%F[IND]		[[ALL] suchbegriff] [IN find-bereich] [alignment]
%M[OVE]		sender INTO empfänger [REP]
%SD[UMP]	SY	[dump-bereich][,...] [medium-u-menge][,...]
%SET		sender INTO empfänger

**Verwaltung**

Kommandoname		Operanden
%AID		[CHECK] [REP] [SYMCHARS] [OV] [LOW] [DELIM] [EXEC] [FORK] [LANG] [EBCDIC] [CCS] [LEV]
%AINT		[aid-mode] [...]
%BASE		[basis]
%D[UMP]F[ILE]		[link [= datei]]
%H[ELP]		[info-ziel] [medium-u-menge][,...]
%OUT		[ziel-kommando [medium-u-menge][,...]]
%OUTFILE		[link [=datei]]
%Q[UALIFY]		[vorqualifikation]
%SYMLIB	SY	[qualifikation-u-lib][,...]
%SHOW		[show-ziel]
%TITLE		[seitenkopf]

### 3.3.1 Ablaufüberwachung

Die Kommandos %CONTROL<sub>n</sub>, %INSERT und %ON dienen der dynamischen Überwachung des Programmablaufs. Mit ihnen vereinbaren Sie Überwachungsbedingungen und Subkommandos (siehe [Kapitel „Subkommando“ auf Seite 51](#)). Tritt die Überwachungsbedingung ein, wird das zugehörige Subkommando bearbeitet. Mit jedem der drei Kommandos legen Sie eine andere Art von Überwachungsbedingung fest, die Sie mit %REMOVE wieder aufheben können.

**%CONTROL<sub>n</sub> *kriterium***

Mit *kriterium* bezeichnen Sie den Typ der zu überwachenden Anweisungen oder Maschinenbefehle.

**%INSERT *testpunkt***

Mit *testpunkt* bezeichnen Sie eine Adresse im ausführbaren Teil des Programms.

**%ON {*write-ereignis* | *ereignis*}**

Mit *write-ereignis* schalten Sie die Schreibüberwachung ein. Mit *ereignis* beschreiben Sie ein Ereignis im Programmablauf, z.B. Adressierungsfehler, Systemaufruf (SVC) oder Nachladen eines Moduls.

**%REMOVE *ziel***

Mit dem Kommando %REMOVE können Sie Überwachungsvereinbarungen wieder aufheben. Mit *ziel* geben Sie an, welche Vereinbarung Sie aufheben.

Je nachdem, was Sie in Ihrem Test überwachen wollen, wählen Sie das entsprechende Kommando aus. Mit dem zugehörigen Subkommando können Sie gezielte Eingriffe in den Programmablauf vornehmen. Mit einem Subkommando vereinbaren Sie ein Kommando oder eine Kommandofolge und eventuell eine Bedingung. Außerdem können Sie dem Subkommando einen Namen geben, mit dem Sie seinen Durchlaufzähler ansprechen oder das Subkommando löschen können. Mit den entsprechenden AID-Kommandos können Sie im Subkommando steuern, ob das Programm unterbrochen werden oder ob es weiterlaufen soll. So ist es möglich, einen automatischen Testablauf vorzubereiten. Allerdings müssen Sie bereits bei der Eingabe des Überwachungskommandos über alle erforderlichen Informationen verfügen, die bei Eintreten der Überwachungsbedingung für das weitere Vorgehen von Bedeutung sind. Sie müssen also nicht mehr mit der Eingabe von Kommandos am Terminal in den Testablauf eingreifen, um z.B. aktuelle Speicher- oder Registerstände zu ändern, wenn Sie die entsprechenden Kommandos im Subkommando hinterlegen.

### 3.3.2 Ablaufsteuerung

Mit den Kommandos %CONTINUE, %RESUME, %STOP und %TRACE verändern Sie den Zustand eines geladenen Programms. Mit %JUMP können Sie für FOR1- und COBOL-Programme eine Fortsetzungsadresse vereinbaren, die vom codierten Programmablauf abweicht. Ein geladenes Programm kann sich in einem von drei Programmzuständen befinden:

1. Das Programm steht.  
%STOP, K2-Taste oder Ende eines %TRACE unterbrechen ein laufendes Programm. 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.

Wenn Sie keine andere Fortsetzungsadresse vereinbart haben, wird der Programmablauf an der Unterbrechungsstelle fortgesetzt. Mit %JUMP (nur FOR1 und COBOL) oder durch Änderung des Befehlszählers (%PC) mit %MOVE oder %SET können Sie eine andere Fortsetzungsadresse festlegen. Für beide Eingriffe in den Programmablauf gilt, dass Sie immer selbst dafür sorgen müssen, dass Speicherinhalte, Registerstände, Datei-Status und -Inhalte auch zur vereinbarten Fortsetzungsadresse passen.

%CONTINUE und %RESUME starten ein geladenes Programm oder setzen es fort. Der Unterschied zwischen den beiden Kommandos besteht darin, dass %RESUME einen noch aktiven %TRACE löscht, %CONTINUE hingegen führt den %TRACE fort.

%STOP hält das Programm an und gibt eine Meldung aus (STOP-Meldung), die Informationen über die aktuelle Unterbrechungsstelle enthält.

%TRACE schaltet die Ablaufverfolgung ein. Das Programm läuft, und die ausgewählten Befehle werden protokolliert. Beendet wird ein %TRACE, wenn die angegebene Anzahl von Befehlen protokolliert ist oder wenn nach einer Unterbrechung das Programm mit %RESUME fortgesetzt wird. Ist der %TRACE nur unterbrochen, weil ein Subkommando ausgeführt wurde, das ein %STOP enthielt, oder ein *steuerungs*-Operand KEEP oder STOP ausgeführt wurde, oder weil die K2-Taste gedrückt wurde, so kann der %TRACE mit %CONTINUE fortgesetzt werden.

Mit dem Operanden *fortsetzung* des Kommandos %TRACE können Sie steuern, ob das Programm nach Beendigung des %TRACE anhält oder ohne Protokollierung weiterläuft.

### 3.3.3 Ausgabe und Modifikation von Speicherinhalten

Mit den Kommandos %DISPLAY, %SDUMP und %DISASSEMBLE können Sie sich Speicherinhalte und Informationen zum Programm ausgeben lassen.

Mit den Kommandos %MOVE und %SET können Sie Speicherinhalte im geladenen Programm verändern.

Mit %FIND können Sie Zeichenfolgen suchen.

%DISPLAY gibt Ihnen den aktuellen Inhalt von Speicherobjekten, deren Adressen bzw. Längen oder die Werte von Konstanten, Anweisungsnamen und Source-Referenzen aus. Sie können mit %DISPLAY auch Systeminformationen abfragen, den Vorschub nach SYSLST steuern oder AID-Literale ausgeben lassen, z.B. um damit den Testverlauf zu kommentieren. Die Ausgabe erfolgt über SYSOUT, SYSLST oder in eine katalogisierte Datei.

Wenn Sie ein Speicherobjekt mit seinem Namen ansprechen, gibt AID es in dem Datentyp und der Länge aus, die im Quellprogramm festgelegt wurde. Mit der Typ- und/oder Längenmodifikation können Sie eine andere Aufbereitung vereinbaren.

%SDUMP gibt einen symbolischen Dump aus. Sie können sich entweder Daten der aktuellen Aufrufhierarchie oder die Aufrufhierarchie selbst ausgeben lassen. Die aktuelle Aufrufhierarchie reicht von der Unterprogrammebene, in der das Programm unterbrochen wurde, über die durch CALL-Anweisungen aufgerufenen Unterprogramme bis zum Hauptprogramm. %SDUMP %NEST gibt die Namen aller Programmteile der aktuellen Aufrufhierarchie aus, soweit AID die Verknüpfungskonventionen bekannt sind. Aus den Programmteilen dieser Hierarchie können Sie dann Daten oder Datenbereiche ausgeben lassen.

Mit %DISASSEMBLE können Sie Speicherinhalte im ausführbaren Teil eines Programms rückübersetzen lassen. AID gibt die Speicherinhalte in symbolischer Assembler-Notation aufbereitet aus. Für Speicherinhalt, der nicht als Befehl interpretiert werden kann, wird eine Ausgabezeile erzeugt, die die sedezimale Darstellung des Speicherinhalts und den Hinweis INVALID OP CODE enthält.

Mit %MOVE verändern Sie Speicherinhalte im geladenen Programm. %MOVE überträgt einen Sender auf einen Empfänger ohne zu überprüfen, ob die Speichertypen von Sender und Empfänger miteinander verträglich sind und ohne Typanpassung. Die Übertragung wird in der Länge von Sender durchgeführt. Sender wird linksbündig in Empfänger übertragen. AID überprüft nur, dass nicht über die rechte Bereichsgrenze (=End-adresse) von Empfänger geschrieben wird.

Zu %MOVE können Sie den Änderungsdialog einschalten oder REP-Sätze erzeugen lassen.

Mit %SET verändern Sie Speicherinhalte im geladenen Programm. %SET überträgt einen Sender auf einen Empfänger und überprüft vor der Übertragung, ob die Speichertypen von Sender und Empfänger miteinander verträglich sind. Die Übertragung wird in der Länge von Empfänger durchgeführt. Sender wird typgerecht in Empfänger übertragen, bei Bedarf

wird abgeschnitten, aufgefüllt oder eine Typ-Anpassung durchgeführt. Die Regeln für die Übertragung mit %SET lehnen sich eng an die jeweilige Programmiersprache an. Die %SET-Beschreibung in den sprachspezifischen Handbüchern enthält eine Tabelle der zulässigen Speichertyp-Kombinationen.

Zu %SET können Sie den Änderungs-Dialog einschalten.

Mit %FIND können Sie in Daten oder im gesamten Benutzer-Adressraum des geladenen Programms bzw. in einer Dump-Datei eine Zeichenfolge suchen und die Treffer auf Terminal (SYSOUT) ausgeben lassen.

Bei einem Treffer gibt AID die Adresse aus, an der die Zeichenfolge gefunden wurde und, wenn möglich, den Namen der zugehörigen CSECT oder des COMMON und die Distanz zur Anfangsadresse von CSECT oder COMMON. Dazu wird der Speicherinhalt ab der Trefferadresse bis zum Ende des Suchbereichs, höchstens aber in der Länge von 12 Bytes, ausgegeben. Zusätzlich speichert AID die Trefferadresse im AID-Register %0G und die Fortsetzungsadresse (Trefferadresse + Suchstringlänge) im AID-Register %1G ab.

### 3.3.4 Verwaltungsfunktionen

Mit den Kommandos %DUMPFIL, %SYMLIB und %OUTFILE verwalten Sie AID-Eingabe- und AID-Ausgabedateien. Sie weisen ihnen Linknamen zu und öffnen oder schließen sie.

Mit %OUT steuern Sie die Ausgaben von AID, mit %TITLE legen Sie eine Kopfzeile für die Ausgabe nach SYSLST fest.

Mit %AID, %AINT, %BASE und %QUALIFY vereinbaren Sie globale Voreinstellungen.

Mit %HELP können Sie sich Hilfe-Texte ausgeben lassen.

Mit %SHOW können Sie sich über die aktuell gültigen Voreinstellungen und über die AID-Kommandos informieren lassen, die Sie im bisherigen Testverlauf eingegeben haben und die noch aktiv sind.

Mit %DUMPFIL verwalten Sie Dump-Dateien. Sie werden zugewiesen über die AID-Linknamen D0 bis D7. Sie können AID veranlassen, eine Dump-Datei zu öffnen oder zu schließen. In den Dump-Dateien befinden sich Speicherabzüge, die Sie in Ihrem Test verwenden wollen.

Mit %SYMLIB können Sie PLAM-Bibliotheken öffnen, in denen Sie die OMs oder LLMs Ihres Programms mit den LSD-Sätzen abgespeichert haben.

Auf geöffnete PLAM-Bibliotheken greift AID zu, wenn Sie in einem Kommando symbolische Namen ansprechen, die in einer Übersetzungseinheit (bei Fortran: Programmeinheit) liegen, für die keine LSD-Sätze geladen wurden. Mit einer Basisqualifikation können Sie die PLAM-Bibliothek einem bestimmten AID-Arbeitsbereich zuordnen.

Bei der Anmeldung mit %SYMLIB stellt AID nur fest, ob die angegebene Bibliothek geöffnet werden kann; AID überprüft nicht, ob der Inhalt einer Bibliothek zu dem Programm passt, das gerade bearbeitet wird. Dadurch ist es möglich, vorbereitend die Bibliotheken anzumel-

den, die Sie während eines Testlaufs eventuell benötigen. Sind für eine Basisqualifikation mehrere Bibliotheken angemeldet, so durchsucht AID sie in der Reihenfolge, in der sie im %SYMLIB-Kommando angegeben wurden.

AID kann bis zu 14 PLAM-Bibliotheken parallel verwalten.

Mit %OUTFILE verwalten Sie AID-Ausgabedateien, in die die Ausgaben der Kommandos %DISASSEMBLE, %DISPLAY, %HELP, %SDUMP und %TRACE oder die REPs des %MOVE-Kommandos geschrieben werden. Sie werden den AID-Linknamen F0 bis F7 zugewiesen. Falls eine Datei noch nicht existiert, wird sie durch AID katalogisiert und geöffnet. Außerdem können Sie veranlassen, dass offene Ausgabedateien wieder geschlossen werden.

Mit %OUT legen Sie für die Ausgabekommandos %DISASSEMBLE, %DISPLAY, %HELP, %SDUMP und %TRACE fest, welche Ausgabemedien verwendet werden und ob Zusatzinformationen ausgegeben werden sollen. Ausgabemedien sind das Terminal (SYSOUT), SYSLST oder eine AID-Ausgabedatei, die zuvor mit %OUTFILE zugewiesen werden kann. Bei den Kommandos %DISPLAY, %HELP und %SDUMP können Sie lokal einen eigenen *medium-u-menge*-Operanden angeben, der für diese Kommandos die Vereinbarungen des %OUT-Kommandos vorübergehend außer Kraft setzt. Die Kommandos %DISASSEMBLE und %TRACE bieten diesen Operanden nicht an.

Mit %TITLE können Sie für die Ausgabe nach SYSLST einen eigenen Seitenkopftext definieren und den Seitenzähler steuern. Die Ausgabe nach SYSLST wird mit %OUT oder dem *medium-u-menge*-Operanden eines Ausgabekommandos vereinbart.

Mit %AID können Sie den Änderungsdialog einschalten, REPs zu den Speicheränderungen mit %MOVE erstellen lassen, die Interpretation des Bindestrichs in Namen steuern, die Überlagerungsstruktur eines Programms (Overlay) berücksichtigen lassen, Klein- und Großbuchstabenunterscheidung aus Benutzereingaben einschalten, andere Begrenzer für Ausgaben vom Typ Character festlegen, einen EBCDIC-Zeichensatz für Konversionen von oder nach UTF16/UTFE oder für die Interpretation und Darstellung von Zeichen festlegen oder auf die englische Ausgabe von Hilfe-Texten umschalten.

Mit %AINT können Sie die Interpretation von indirekten Adressangaben in AID-Kommandos umschalten. Sie legen damit fest, ob AID eine Adresse vor einem Pointer-Operator

(->) als 24-Bit-Adresse oder als 31-Bit-Adresse interpretieren soll. Der Adressierungsmodus des Testobjekts wird damit nicht beeinflusst. Mit einer Basisqualifikation können Sie den Bereich vereinbaren, für den die Angabe zur Adressinterpretation gelten soll.

Mit %BASE vereinbaren Sie die Basisqualifikation. Sie gilt für alle nachfolgenden Kommandos, in denen Sie keine explizite Basisqualifikation angeben. Mit %BASE legen Sie fest, ob sich der AID-Arbeitsbereich im geladenen Programm oder in einer Dump-Datei befinden soll. Wenn Sie als Basisqualifikation eine Dump-Datei angeben, muss diese zuvor mit %DUMPFILe zugewiesen werden.

Mit %QUALIFY bezeichnen Sie Qualifikationen oder eine Adresse, auf die Sie sich im Adressoperanden eines anderen Kommandos durch Voran Stellen eines Punktes beziehen möchten. Diese verkürzte Schreibweise ist immer dann sinnvoll, wenn Sie mehrfach Adressen ansprechen wollen, die die gleichen Qualifikationen erfordern oder die von der gleichen Ausgangsadresse aus berechnet werden.

Mit %HELP können Sie sich über die AID-Kommandos informieren. Auf das gewählte Ausgabemedium werden ausgegeben: entweder alle AID-Kommandos oder das gewählte Kommando und seine Operanden. Der HELP-Informationen-Pool umfasst Informationen für das symbolische Testen sowie für das Testen auf Maschinencode-Ebene.

Mit %SHOW können Sie sich über die AID-Kommandos des bisherigen Testverlaufs, über die aktuell gültigen globalen Einstellungen oder über die aktuell gültigen Operandenwerte der Kommandos informieren.

%SHOW ohne Operand zeigt das zuletzt eingegebene AID-Kommando an.

Zu den Kommandos der Ablaufüberwachung (%CONTROL<sub>n</sub>, %INSERT und %ON) gibt %SHOW eine Liste der Original-Eingabestrings aller aktiven %CONTROL<sub>n</sub> oder %ON oder die Liste der gesetzten Testpunkte mit Kontext, virtueller Adresse, CSECT oder COMMON und Distanz zum CSECT- oder COMMON-Anfang aus. Den Original-Eingabestring zu einem bestimmten Testpunkt erhalten Sie mit %SHOW %INSERT *testpunkt*. Sind am Testpunkt mehrere %INSERTs aktiv, so werden die Kommandos in der umgekehrten Reihenfolge ausgegeben, d.h. der zuletzt eingegebene %INSERT wird als erster aufgelistet.

Zu %SHOW %TRACE gibt AID das zuletzt eingegebene %TRACE-Kommando, die abgearbeiteten %TRACE-Schritte und die aktuell gültigen Operandenwerte aus.

Zu %SHOW %DISASSEMBLE erhalten Sie die aktuell gültigen Operandenwerte, zu

%SHOW %FIND das zuletzt eingegebene Kommando und den letzten Treffer.

Zu den Kommandos, die AID-Ein- oder Ausgabedateien verwalten, gibt %SHOW alle explizit oder implizit geöffneten Dateien sowie verschiedene Zusatzinformationen aus.

%SHOW %OUT zeigt die aktuellen Ausgabe-Vereinbarungen der Kommandos an, deren Ausgabe über %OUT gesteuert wird. Zu %SHOW %AID, %SHOW %BASE oder %SHOW %QUALIFY werden die mit diesen Kommandos getroffenen Vereinbarungen ausgegeben.

### 3.3.5 Übersicht über die Geltungsdauer der Kommandos

Beim Arbeiten mit AID ist es wichtig zu wissen, welche Kommandos, Operandenwerte oder Vereinbarungen bis zur Eingabe des nächsten Kommandos gleichen Typs, bis zum Ende des Programms oder Tasks gültig sind. Dazu dient die folgende Übersicht:

Kommando	Operand	Gültigkeitsdauer
%AID	alle	gültig bis zur Eingabe eines neuen %AID mit entsprechendem Operanden oder bis /EXIT-JOB.



Kommando	Operand	Gültigkeitsdauer
%AINT	aid-mode	gültig bis zur Eingabe eines neuen %AINT zur selben Basisqualifikation oder bis /EXIT-JOB oder bis Schließen der zugehörigen Dump-Datei.
%BASE	basis	gültig bis zur Eingabe eines neuen %BASE oder bis /EXIT-JOB oder bis zum Schließen der als basis vereinbarten Dump-Datei.
%CONTROLn	kriterium/ control- bereich	kann vom nächsten %CONTROLn übernommen werden, ansonsten gültig bis zum Löschen des %CONTROLn oder Programmende.
	subkdo	muss stets angegeben werden.
%DISASSEMBLE	anzahl	kann von einem neuen %DISASSEMBLE übernommen werden; diese Möglichkeit besteht bis Programmende.
	start	Die an den zuletzt rückübersetzten Befehl anschließende Adresse kann als start-Wert übernommen werden; diese Möglichkeit besteht bis Programmende.
%DISPLAY	alle	Alle Operanden müssen stets angegeben werden.
%DUMPFILe	link=datei	Die über link zugewiesene Datei bleibt geöffnet bis /EXIT-JOB, falls sie nicht explizit geschlossen wird.
%FIND	suchbe- griff	kann aus einem vorherigen %FIND übernommen werden, bis find-bereich vollends durchsucht ist. Diese Möglichkeit besteht bis /EXIT-JOB.
	find- bereich/ alignment	Ist kein suchbegriff angegeben, wird find-bereich/alignment aus dem vorherigen %FIND übernommen, bis find-bereich vollends durchsucht ist.
%INSERT	testpunkt	Der Testpunkt bleibt eingetragen, bis er mit %REMOVE gelöscht wurde, bis alle %INSERTs gelöscht wurden oder Programmende. Bei Programmen, die als Overlays gebunden sind, bleibt der Testpunkt im Modul eingetragen, in dem er gesetzt wurde, auch wenn an gleicher Stelle inzwischen ein anderer Modul geladen ist.
	subkdo	Das Subkommando bleibt eingetragen, bis es mit %REMOVE gelöscht wird, bis der zugehörige Testpunkt gelöscht wird oder bis Programmende.
%MOVE	alle	Alle Operanden müssen stets angegeben werden.

Kommando	Operand	Gültigkeitsdauer
%ON	ereignis/ write- ereignis	Das Ereignis bleibt eingetragen, bis es mit %REMOVE gelöscht wurde, bis alle %ON gelöscht werden oder bis Programmende. Eine Ausnahme bildet %ON %WRITE: ein neues write-Ereignis überschreibt ein schon eingetragenes Ereignis.
	subkdo	Das Subkommando bleibt eingetragen, bis es mit %REMOVE gelöscht wird, bis das zugehörige ereignis gelöscht wird, bis alle %ON gelöscht werden oder bis Programmende.
%OUT	alle	gültig bis zur Eingabe eines neuen %OUT mit entsprechendem Operanden oder bis /EXIT-JOB.
%OUTFILE	link=datei	Wenn Datei nicht explizit geschlossen wird, bleibt sie geöffnet bis /EXIT-JOB.
%QUALIFY	vorquali- fikation	Vorqualifikation gilt, bis sie durch einen neuen %QUALIFY überschrieben wird, durch einen %QUALIFY ohne Operanden aufgehoben wird oder bis /EXIT-JOB.
%REMOVE	ziel	ziel muss stets angegeben werden.
%SDUMP	alle	Es kann nichts aus einem vorherigen %SDUMP übernommen werden.
%SET	alle	Alle Operanden müssen stets angegeben werden.
%SHOW	info-ziel	Es kann nichts aus einem vorherigen %SHOW übernommen werden.
%SYMLIB	qual-u-lib	Eine Bibliothek bleibt angemeldet bis zum nächsten %SYMLIB zur selben Basisqualifikation, bis zum nächsten %SYMLIB ohne Operanden, bis /EXIT-JOB oder bis zum Schließen der zugehörigen Dump-Datei.
%TITLE	seitenkopf	gültig bis zum nächsten %TITLE oder Programmende.
%TRACE	alle	Alle Operanden gelten solange, bis sie durch entsprechende Angaben in einem neuen %TRACE überschrieben werden oder bis Programmende. trace-bereich wird nicht übernommen, wenn ein %TRACE ohne Trace-Bereich eingegeben wird und die Unterbrechungsstelle nicht im Trace-Bereich liegt.

---

## 4 Voraussetzungen zum Testen mit AID

Beim Testen mit AID wird unterschieden zwischen dem symbolischen Testen, wobei die im Quellprogramm vergebenen Namen zur Adressierung verwendet werden und dem maschinennahen Testen, wenn mit virtuellen Adressen gearbeitet wird. Um symbolisch testen zu können, müssen Sie beim Übersetzen den Compiler veranlassen, LSD-Sätze zu generieren. Beim Binden und Laden können Sie über Operanden steuern, dass die LSD-Sätze mit eingebunden bzw. mitgeladen werden. Wenn jedoch die LSD-Sätze beim Binden und Laden nicht berücksichtigt wurden, können Sie sie von AID aus einer PLAM-Bibliothek nachladen lassen.

Zum Testen auf Maschinencode-Ebene bedarf es dagegen keiner besonderen Maßnahmen.

Wenn CSECTs mit der LMS-Anweisung MODIFY-ELEMENT (Subanweisung RENAME-SYMBOLS) umbenannt werden, kann nicht mehr symbolisch getestet werden. Wurde ein LLM vom Compiler direkt erzeugt und enthält der LLM die LSD-Sätze, können Sie CSECTs mit BINDER umbenennen (Anweisung MODIFY-MODULE-ATTRIBUTES). Für welche Compiler-Versionen diese Möglichkeit besteht, können Sie im Benutzerhandbuch des jeweiligen Compilers nachlesen.

Das maschinennahe Testen wird durch das Umbenennen von CSECTs nicht beeinträchtigt. Die CSECTs Ihres Programms können Sie mit den neuen Namen ansprechen.

## 4.1 Testen auf Maschinencode-Ebene

Für das Testen im Maschinencode brauchen Sie beim Übersetzen, Binden und Laden keine speziellen Operanden angeben, um später alle Funktionen ausnutzen zu können, wie sie im Handbuch für das „[Testen auf Maschinencode-Ebene](#)“ [1] beschrieben sind.

Beim Binden wird standardmäßig die Objekt-Strukturliste aus dem Verzeichnis der Externbezüge (siehe „[Bindelader-Starter in BS2000](#)“ [13]) erzeugt. Dies ist beim BINDER das ESV (External Symbols Vector).

Keine Objekt-Strukturliste wird jedoch erzeugt, wenn Sie beim Binden eines Programms mit BINDER in der Anweisung SAVE-LLM den Operanden SYMBOL-DICTIONARY=NO schreiben. Dann können Sie die folgenden Funktionen nicht ausführen:

- Ausgeben einer Liste aller CSECTs und COMMONs des Benutzerprogramms (%D %SORTEDMAP oder %D %MAP)
- Ausgeben der Lokalisierungsinformationen einer Speicherreferenz (%D %LOC(speicherref))
- Angeben einer CSECT-/COMMON-Qualifikation in einer Speicherreferenz
- Ablaufüberwachung durch die Kommandos %CONTROL<sub>n</sub> und %TRACE, wenn sie implizit oder explizit auf eine CSECT beschränkt sein sollen
- Erzeugen von REPs für Änderungen

Außerdem kann AID in diesem Fall bei %TRACE, %DISASSEMBLE, %FIND und in der STOP-Meldung keine CSECT-/COMMON-relativen Adressen ausgeben.



Vorsicht bei LLMs oder Kontexten, die gleichnamige CSECTs enthalten: in diesem Fall ist nicht vorhersehbar, welche CSECT mit AID angesprochen wird.

## 4.2 Symbolisches Testen

Beim symbolischen Testen mit AID können Sie Daten mit den von Ihnen definierten Namen aus dem Quellprogramm ansprechen oder sich auf Anweisungen durch die Angabe von Anweisungsnamen oder Source-Referenzen beziehen. Dafür benötigt AID Informationen über die verwendeten Namen aus den Quellprogrammen, aus denen das zu testende Programm entstanden ist. Diese Information besteht aus zwei Teilen:

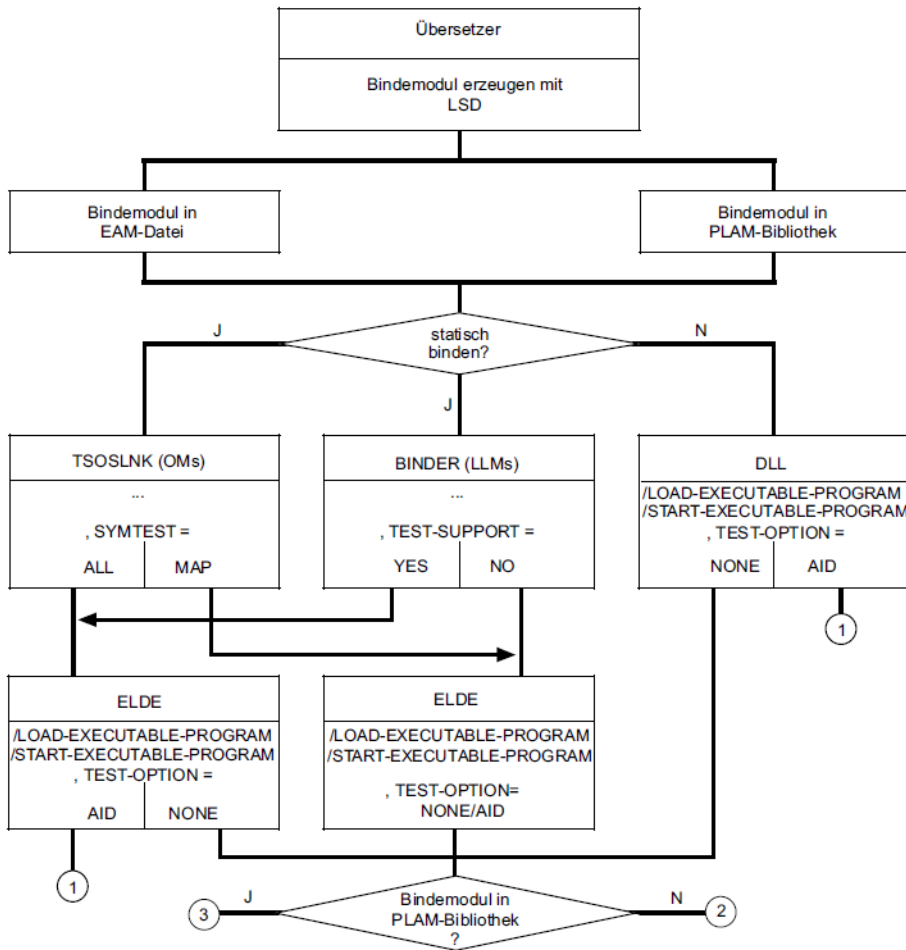
1. LSD (List for Symbolic Debugging), Verzeichnis der im Modul definierten Namen und Source-Referenzen.
2. ESD (External Symbolic Dictionary), ESV (External Symbols Vector) beim Binden mit BINDER.

In den folgenden Abschnitten werden die verschiedenen Möglichkeiten, ESD/ESV- und LSD-Sätze zu behandeln, für jeden der folgenden Schritte im Programm-Entwicklungsprozess beschrieben:

- Übersetzen des Quellprogramms
- Binden und Laden mit DBL oder
- Binden mit BINDER

Außerdem bietet AID noch das Kommando %SYMLIB, mit dem PLAM-Bibliotheken (siehe Handbuch „LMS (BS2000)“ [11]) geöffnet werden, aus denen AID bei Bedarf die fehlenden LSD-Sätze nachlädt.

Die folgende Graphik gibt einen Überblick über die Möglichkeiten, die vom Compiler erzeugten LSD-Sätze beim Binden und Laden mitzuführen oder nicht.



- (1) Das Programm kann uneingeschränkt symbolisch getestet werden.
- (2) Das Programm kann nur eingeschränkt symbolisch getestet werden, d.h. Namen von Programmteilen können angesprochen und Aufrufhierarchien rückverfolgt werden.
- (3) Das Programm kann erst symbolisch getestet werden, wenn mit dem Kommando %SYMLIB die PLAM-Bibliothek mit den OMs oder LLMs zugewiesen wurde.

Es gibt also verschiedene Wege, wie AID die LSD-Informationen erhält. Voraussetzung ist immer, dass beim Übersetzen die LSD-Sätze an den erzeugten Bindemodul (OM) oder Bindelademodul (LLM) übergeben werden. Speichern Sie den OM oder LLM in einer PLAM-Bibliothek ab, können Sie die LSD-Sätze entweder beim Binden und Laden einbeziehen oder sie erst bei Bedarf durch AID nachladen lassen.

Das Nachladen der LSD-Sätze ist besonders für Programme sinnvoll, die aus mehreren Übersetzungsläufen entstanden sind und von denen nur einzelne Module symbolisch getestet werden sollen. Die nachzuladende LSD muss im selben Übersetzungslauf erzeugt worden sein wie der Modul.

Aus der temporären Bindemodul-Datei (\*OMF-Datei) kann AID nicht nachladen.

### 4.2.1 Übersetzen

Über eine Compiler-Option steuern Sie die Erzeugung der LSD-Sätze durch den Compiler. Die genaue Beschreibung der Operanden finden Sie in den sprachspezifischen Handbüchern zu AID (siehe [2] - [6]). Prinzipiell wird zwischen zwei Möglichkeiten unterschieden:

- Es werden nur ESD/ESV-Sätze, aber keine LSD-Sätze erzeugt (Standardwert). Das Programm kann nur maschinennah getestet werden.
- Vom Compiler werden ESD/ESV- und LSD-Sätze erzeugt. Das Programm kann mit AID symbolisch getestet werden.

### 4.2.2 Binden mit BINDER

Beim Binden mit BINDER können bei allen Bindevorgängen LSD-Sätze miteingebunden werden.

In den BINDER-Anweisungen, die das Anlegen, Ändern oder Abspeichern eines LLM steuern, bestimmt der Operand TEST-SUPPORT, ob die LSD-Sätze aus Bindelademodulen (LLMs) miteingebunden werden oder nicht (siehe Handbuch „Binder in BS2000“ [12]). Anweisungen, die die gleichen Operandenwerte verlangen, werden in der folgenden Tabelle zusammengefasst und mit der Syntax des TEST-SUPPORT-Operanden aufgeführt. Die Beschreibung der Operandenwerte schließt sich an die Tabelle der Anweisungen und der zugehörigen TEST-SUPPORT-Syntax an.

Anweisung	Bedeutung	TEST-SUPPORT-Operandenwerte
START-LLM-CREATION	Erzeugen eines LLM	TEST-SUPPORT= { *YES *NO }
START-LLM-UPDATE MODIFY-LLM- ATTRIBUTES	Ändern eines LLM Ändern der physischen Struktur eines LLM	TEST-SUPPORT= { *UNCHANGED *YES *NO }
MODIFY-MODULE- ATTRIBUTES	Ändern der logischen Struktur eines LLM	TEST-SUPPORT= { *UNCHANGED *INCLUSION-DEFAULT *YES *NO }
SAVE-LLM	Speichern eines LLM	TEST-SUPPORT= { *LAST-SAVE *YES *NO }
INCLUDE-MODULES REPLACE-MODULES RESOLVE-BY-AUTO- LINK	Einfügen von Modulen Ersetzen von Modulen Befriedigen von Ex- ternverweisen durch Autolink	TEST-SUPPORT= { *INCLUSION-DEFAULT *YES *NO }

\*YES

Die LSD-Sätze werden übernommen. Der Binder überprüft jedoch nicht, ob der verarbeitete Bindemodul (OM) oder Bindelademodul (LLM) tatsächlich LSD-Sätze enthält.

\*NO

Die LSD-Sätze werden nicht übernommen. Falls in der SAVE-LLM-Anweisung jedoch SYMBOL-DICTIONARY=YES gesetzt wurde, d.h. das ESV wurde mitgeführt, können Aufrufhierarchien rückverfolgt werden. Sind zusätzlich bei der Übersetzung LSD-Sätze erzeugt und mit dem OM oder LLM in eine PLAM-Bibliothek geschrieben worden, können Sie zum symbolischen Testen bei Bedarf die LSD nachladen.

\*UNCHANGED

Die derzeit geltende Einstellung, die von den zuvor eingegebenen Kommandos abhängt, wird beibehalten.



**\*INCLUSION-DEFAULT**

Es werden die Werte des Operanden TEST-SUPPORT, der ein Unteroperand des Operanden INCLUSION-DEFAULTS ist, aus den Anweisungen START-LLM-CREATION, START-LLM-UPDATE, oder MODIFY-LLM-ATTRIBUTES desselben Edit-Laufs übernommen.

**\*LAST-SAVE**

Der Binder übernimmt die Werte aus der letzten SAVE-LLM-Anweisung desselben Edit-Laufs. Wurde bisher noch kein SAVE-LLM angegeben, setzt der Binder YES ein.



Der BINDER erlaubt, gleichnamige CSECTs mehrfach in einen LLM einzubinden. Beim Testen mit AID führt dies jedoch zu unvorhersehbaren Ergebnissen.

### 4.2.3 Binden und Laden mit DBL

Ein Programm, das getestet werden soll, wird mit dem BS2000-Kommando LOAD-EXECUTABLE-PROGRAM aufgerufen, damit anschließend die AID-Kommandos eingegeben werden können. Ein Programm, das nur bei Auftreten eines Fehlers noch mit AID bearbeitet werden soll, kann mit START-EXECUTABLE-PROGRAM geladen und gestartet werden. Auch eine mit dem Makroaufruf BIND eingebundene Ladeeinheit kann mit AID getestet werden.

Ein Programm in Form von Bindemodulen (OMs) oder Bindelademodulen (LLMs) wird durch den dynamischen Bindelader DBL geladen (siehe Handbuch „[Bindelader-Starter in BS2000](#)“ [13]).

Laden bzw. Laden und Starten mit dem DBL aufgerufen durch SDF-Kommandos:

```

-----
{ /LOAD-EXECUTABLE-PROGRAM } ..... ,TEST-OPTIONS = { *DBL-DEFAULT }
{ /START-EXECUTABLE-PROGRAM }                       { *NONE
                                                        { *AID
                                                        }
-----

```

**\*DBL-DEFAULT**

Der Operandenwert wird aus dem letzten Aufruf des Kommandos MODIFY-DBL-DEFAULTS übernommen. Falls für den betreffenden Operanden mit MODIFY-DBL-DEFAULTS noch kein Wert festgelegt wurde, gilt TEST-OPTIONS=\*NONE.

**\*NONE**

Das Programm wird ohne LSD-Sätze geladen. Symbolisches Testen ist nur möglich, wenn AID zum Nachladen der LSD-Sätze die PLAM-Bibliothek zur Verfügung gestellt wird, in der die zugehörigen OMs oder LLMs stehen.

**\*AID** Das Programm wird mit den LSD-Sätzen geladen. Es wird auch dann geladen, wenn es keine LSD-Sätze enthält. Wenn gleichzeitig DBL-PARAMETERS:LOADING angegeben wurde, ist darauf zu achten, dass der zugehörige Operand LOAD-INFORMATION, der das Laden des ESV steuert, auf DEFINITIONS (Standardwert) oder auf REFERENCES gesetzt ist.

- Einbinden einer weiteren Ladeeinheit mit dem DBL über den Makroaufruf BIND:

```
-----
BIND      . . . . . ,TSTOPT = { *DBLOPT
                             NONE
                             AID }
-----
```

**\*DBLOPT** Der Operandenwert wird aus dem letzten Aufruf des Kommandos MODIFY-DBL-DEFAULTS übernommen. Falls für den betreffenden Operanden mit MODIFY-DBL-DEFAULTS noch kein Wert festgelegt wurde, gilt TSTOPT=NONE.

**NONE** Bedeutung wie oben

**AID** Das Programm wird mit den LSD-Sätzen geladen. Es wird auch dann geladen, wenn es keine LSD-Sätze enthält. Gleichzeitig muss der Operand LDINFO auf DEF oder REF gesetzt sein, damit das ESV geladen wird.

## Beispiele

1. /LOAD-EXECUTABLE-PROGRAM FROM-FILE=\*OMF,TEST-OPTIONS=\*AID

Aus der \*OMF-Datei lädt der dynamische Bindelader einen Bindemodul mit LSD-Sätzen.

2. /LOAD-EXECUTABLE-PROGRAM FROM-FILE=\*LIBRARY-ELEMENT(LIBRARY=PROGRAMLIB,ELEMENT-OR-SYMBOL=ROOTMOD)

Aus der PLAM-Bibliothek PROGRAMLIB wird ROOTMOD geladen.

Die Beispiele gelten entsprechend für das START-EXECUTABLE-PROGRAM-Kommando.

## 4.2.4 Nachladen von LSD-Sätzen durch AID

AID kann aus PLAM-Bibliotheken LSD-Sätze für ein Programm nachladen, wenn in der Bibliothek die zugehörigen OMs oder LLMs mit den LSD-Sätzen stehen. Mit dem Kommando %SYMLIB veranlassen Sie AID, die angegebene Bibliothek zu öffnen. Stellt AID bei der Bearbeitung eines Kommandos mit symbolischen Operanden fest, dass die zugehörigen LSD-Sätze nicht im Speicher zur Verfügung stehen, greift es auf mit %SYMLIB zugewiesene und geöffnete Bibliotheken zurück. AID überprüft, ob die nachgeladenen LSD-Sätze aus derselben Übersetzung stammen wie der Modul, zu dem sie nachgeladen werden.

Ist keine Bibliothek zugewiesen oder enthalten die zugewiesenen Bibliotheken nicht den gesuchten OM oder LLM oder sind darin keine LSD-Sätze enthalten, meldet AID, dass die LSD-Sätze fehlen. Die erforderliche Bibliothek können Sie mit einem neuen %SYMLIB zuweisen. Wiederholen Sie dann das AID-Kommando, für das die LSD-Sätze fehlten, kann es von AID bearbeitet werden.

Bei LLMs ist es möglich, darin enthaltene CSECTs mit der BINDER-Anweisung MODIFY-SYMBOL-VISIBILITY zu maskieren. Zu solchen CSECTs kann AID keine LSD-Informationen nachladen. Das gleiche gilt für CSECTs, für die der Operand RUN-TIME-VISIBILITY auf YES gesetzt wurde, da dies die Maskierung der CSECT einschließt. Dieser Operand kann bei den folgenden BINDER-Anweisungen angegeben werden:

- INCLUDE-MODULES
- MODIFY-MODULE-ATTRIBUTES
- REPLACE-MODULES
- RESOLVE-BY-AUTOLINK

Programme, die maskierte CSECTs enthalten, können mit AID nur dann symbolisch getestet werden, wenn die LSD gleich zusammen mit dem Programm geladen wird. Nachladen ist nur möglich, wenn in einem gesonderten BINDER-Lauf die Maskierungen zurückgesetzt wurden.



Es wird darauf hingewiesen, dass AID die LSD-Suche in einem LLM abbricht, wenn es die erste CSECT des gesuchten Namens findet, selbst wenn die LSD zu dieser CSECT inkonsistent ist, weil z.B. die LSD nicht aus der gleichen Übersetzung hervorgegangen ist wie die CSECT. Es nützt also nichts, wenn im selben LLM eine weitere CSECT gleichen Namens mit konsistenter LSD enthalten ist.

## Beispiele

1. `/LOAD-EXECUTABLE-PROGRAM FROM-FILE=*LIBRARY-ELEMENT(LIBRARY=PROGRAMLIB,ELEMENT-OR-SYMBOL=ROOTMOD)`

Das gebundene FORTRAN-Programm ROOTMOD wird ohne LSD-Sätze aus der PLAM-Bibliothek PROGRAMLIB geladen.

Wenn Sie nun das folgende AID-Kommando eingeben, mit dem Sie die Anweisung mit der Anweisungsmarke 10 ansprechen:

```
%INSERT L'10'
```

erhalten Sie die Fehlermeldung:

```
AID0378 Symbolinformation fehlt
```

Enthält die Bibliothek PROGRAMLIB den zum Programm gehörenden Bindemodul mit den LSD-Sätzen, können Sie mit dem Kommando

```
%SYMLIB PROGRAMLIB
```

die erforderliche PLAM-Bibliothek zuweisen und dann das %INSERT-Kommando wiederholen, das nun von AID bearbeitet werden kann. Bindemodul und Ladeeinheit können in verschiedenen Bibliotheken stehen, aber die Ladeeinheit muss aus der Bindemodul-Version gebunden worden sein, aus der die LSD-Sätze nachgeladen werden. Im Fehlerfall gibt AID die folgende Meldung aus:

```
AID0377 Inkonsistente Symbolinformation fuer (&00)
          &00 = Programmname
```

2. `%SYMLIB E=D1.BINDEMODUL.LIB1,E=D1.BINDEMODUL.LIB2`

Für die Dump-Datei mit dem Linknamen D1 stehen die zwei PLAM-Bibliotheken BINDEMODUL.LIB1 und BINDEMODUL.LIB2 zum Nachladen der LSD-Sätze zur Verfügung.

3. `%QUALIFY E=D2`  
`%SYMLIB E=D3.BIB1,.BIB2,BIB3`

Für die Dump-Datei mit dem Linknamen D3 wird die PLAM-Bibliothek BIB1 angemeldet und geöffnet.

Für die Dump-Datei mit dem Linknamen D2 wird die Bibliothek BIB2 angemeldet und geöffnet.

Die PLAM-Bibliothek BIB3 wird für den aktuellen AID-Arbeitsbereich angemeldet und geöffnet.

Wurde bisher kein %BASE-Kommando eingegeben, ist der aktuelle AID-Arbeitsbereich immer der virtuelle Speicherbereich des geladenen Programms. Nach einem %BASE-Kommando ist der AID-Arbeitsbereich der mit %BASE bezeichnete Bereich.

---

# 5 Kommandoeingabe

## 5.1 Kommandoformat

Jedes AID-Kommando beginnt mit dem Prozentzeichen "%", unmittelbar gefolgt vom Kommandonamen.

Nach mindestens einem Leerzeichen können Operanden folgen.

Folgen Operanden und/oder Schlüsselwörter unmittelbar aufeinander ohne ein vorgegebenes Trennzeichen, dann muss mindestens ein Leerzeichen eingefügt werden.

Die Operanden müssen Sie in der Reihenfolge angeben, in der diese im Format aufgeführt sind.

### Namen für Kommandos

Ein AID-Kommando kann wie ein BS2000-Kommando mit einem Namen versehen werden:

- 1. Zeichen: A-Z, \$, # oder @
- alle folgenden Zeichen: A-Z, 0-9, \$, #, @ oder -, wobei der Bindestrich "-" nicht letztes Zeichen des Namens sein darf.

Namen, die mit SKIP-COMMANDS angesprungen werden, beginnen mit einem Punkt und dürfen maximal 8 Zeichen lang sein; Namen aus S-Prozeduren, zu denen mit GOTO verzweigt wird, dürfen maximal 255 Zeichen lang sein und werden mit einem Doppelpunkt abgeschlossen.

Der Name folgt auf den Schrägstrich, der im Dialog vom System vorgegeben oder in Prozedurdateien von Ihnen eingegeben wird. Zwischen dem Namen und dem %-Zeichen des AID-Kommandos muss mindestens ein Leerzeichen stehen.

**Beispiel:** /.START %AID CHECK=NO

Die BS2000-Kommandonamen dienen als Sprungziele in Prozeduren; für das Testen mit AID sind sie jedoch ohne Bedeutung.

## Fortsetzung von Eingabezeilen

Wenn Sie ein AID-Kommando in die nächste Zeile fortsetzen müssen, so gilt dafür derselbe Fortsetzungsmechanismus, wie er auch für BS2000-Kommandos vorgesehen ist. Im Dialog können Sie eine Eingabe einfach über mehrere Zeilen hinwegschreiben; Sie haben aber auch die Möglichkeit, die Zeile mit einem Bindestrich abzuschließen und abzuschicken. Dann beginnt die Folgezeile nach der vom System gesendeten Eingabeaufforderung.

In Prozedurdateien muss die Folgezeile durch einen Bindestrich angekündigt werden, dem nur noch Leerzeichen bis zum Zeilenende folgen dürfen. Die Folgezeile muss mit einem Schrägstrich beginnen.

Ein AID-Kommando darf nicht länger als 1000 Zeichen sein. Da im Speicher nur ein begrenzter Bereich für die Interpretation eines Kommandos zur Verfügung steht, ist die Anzahl der Operanden in einem Kommando beschränkt. Wieviele Operanden jeweils angegeben werden können, finden Sie in der Beschreibung der einzelnen Kommandos.

## Verwendung von Leerzeichen und Kommentaren

Um Ihre AID-Kommandos übersichtlicher und verständlicher zu gestalten, können Sie Leerzeichen und Kommentare einsetzen. Kommentare müssen wie in der BS2000-Kommandosprache zwischen Anführungszeichen (") gesetzt werden. Leerzeichen und Kommentare können Sie überall dort einfügen, wo eins der folgenden Zeichen steht:

␣	Leerzeichen
.	Punkt
,	Komma
=	Gleichheitszeichen
'...'	Hochkomma
(...)	öffnende und schließende runde Klammern
<...>	öffnende und schließende spitze Klammern
[...]	öffnende und schließende eckige Klammern
;	Semikolon
+ - * /	arithmetische Operatoren
->	Pointer-Operator

Bei der Verwendung des Minuszeichens/Bindestrichs "-" ist die Voreinstellung des Operanden *SYMCHARS* im Kommando %AID zu berücksichtigen.

## Beispiel

```
%CONTROL1      %CALL      " SORT-AUFRUF "      <%DISPLAY 'CALL'; %STOP>
```

## 5.2 Einzelkommandos

AID-Kommandos können Sie im BS2000-Kommandomodus eingeben oder auch über die CMD-Makro-Schnittstelle aufrufen. AID-Kommandos werden wie BS2000-Kommandos zuerst vom BS2000 übernommen, anhand des %-Zeichens als AID-Kommandos identifiziert und dann an AID übergeben. Stellt der BS2000-Kommandointerpreter bei der Eingabe des AID-Kommandos fest, dass das Kommando zu lang ist, wird es mit einer Fehlermeldung abgewiesen, und Sie können das korrigierte Kommando erneut eingeben.

AID überprüft Syntax und Semantik eines Kommandos und stellt fest, ob die Operandenwerte in der aktuellen Testsituation bearbeitet werden können. Es führt z.B. zu einer Fehlermeldung, wenn Sie eine symbolische Adresse ansprechen wollen, die in den verfügbaren LSD-Sätzen (siehe [Abschnitt „Grundlegende Begriffe“ auf Seite 20](#)) nicht verzeichnet ist.

Ist ein Kommando syntaktisch falsch, schreibt AID eine entsprechende Fehlermeldung und markiert die Stelle, an der es den Fehler entdeckt hat. Anschließend können Sie das korrigierte Kommando erneut eingeben.

Hat AID ein Kommando übernommen und ausgeführt, dann hängt es vom Typ des Kommandos ab, ob das Programm gestartet wird oder ob Sie weitere Kommandos eingeben können.

## 5.3 Kommandofolgen und Subkommandos

In Kommandofolgen können Sie mehrere AID- und BS2000-Kommandos zusammenfassen. Aufeinanderfolgende Kommandos müssen Sie durch Semikolon trennen. Auch eine Kommandofolge darf nicht länger sein als 1000 Zeichen.

Kommandofolgen werden unmittelbar ausgeführt. Sie werden von links nach rechts abgearbeitet.

Alle Kommandos einer Kommandofolge, die mit % beginnen, werden von AID als AID-Kommandos identifiziert und sofort auf Fehler untersucht. Erkennt AID einen Syntaxfehler, so wird die gesamte Kommandofolge schon bei der Eingabe abgewiesen. Kommandos ohne führendes Prozentzeichen interpretiert AID als BS2000-Kommandos und übernimmt sie ohne weitere Prüfung. Fehlerhafte oder unzulässige BS2000-Kommandos werden daher erst bei der Ausführung erkannt und führen zum Abbruch der Kommandofolge. Auch bei schwerwiegenden Fehlern in AID-Kommandos, wie z.B. Adressenüberlauf, wird die Bearbeitung der Kommandofolge abgebrochen. Danach befinden Sie sich wieder im Kommandomodus und können weitere Kommandos eingeben.

Kann ein AID-Kommando nicht ausgeführt werden, weil ein angegebener Name nicht in den LSD-Sätzen verzeichnet ist oder weil keine LSD-Sätze geladen sind, so gibt AID für dieses Kommando eine entsprechende Fehlermeldung aus und fährt in der Bearbeitung eventuell noch folgender Kommandos fort.

Da nach bestimmten Fehlern die gesamte Kommandofolge erneut eingegeben werden muss, bietet sich die Verwendung längerer Kommandofolgen nur für ausgetestete Prozedurdateien an.

In Kommandofolgen sind alle BS2000-Kommandos zugelassen, die Sie auch im CMD-Makro (siehe Handbuch „[Makroaufrufe an den Ablaufteil](#)“ [10]) angeben können und fast alle AID-Kommandos.

Nicht zugelassen sind dagegen die folgenden Kommandos:

AID-Kommandos:     %AID, %ALIAS, %BASE, %DUMPFIL, %HELP, %OUT,  
                          %QUALIFY, %?

BS2000-Kommandos: Die Liste der Kommandos finden Sie im Anhang.

Auch SDF-P-Kontrollflusskommandos dürfen in Kommandofolgen nicht verwendet werden. Außerdem ist bei der Verwendung einiger BS2000-Kommandos zu berücksichtigen, dass diese zwar in Kommandofolgen erlaubt sind, dass sie aber ein geladenes Programm abbrechen und Sie danach das Programm nicht mehr mit AID-Kommandos bearbeiten können (siehe Beschreibung des Makroaufrufs CMD im Handbuch „[Makroaufrufe an den Ablaufteil](#)“ [10]).

Die Kommandos %TRACE, %RESUME, %CONTINUE oder %STOP brechen eine Kommandofolge ab. Während Sie nach einem %STOP-Kommando wieder im Kommandomodus sind, wird durch die Kommandos %TRACE, %RESUME und %CONTINUE das Programm gestartet bzw. fortgesetzt. Diese Kommandos sind also nur als letztes Kommando einer Kommandofolge sinnvoll.

Ein Subkommando ist kein eigenständiges Kommando, sondern ein Operand der Überwachungskommandos %CONTROL<sub>n</sub>, %INSERT und %ON. Das Subkommando wird erst dann bearbeitet, wenn die Überwachungsbedingung eingetreten ist.

Von folgenden Besonderheiten abgesehen gelten für den Kommandoteil der Subkommandos dieselben Regeln wie für Kommandofolgen:

- Das Überwachungskommando mit dem Subkommando zusammen darf nicht länger sein als 1000 Zeichen.
- Zusätzlich zu %CONTINUE, %RESUME, %STOP und %TRACE ist auch ein %REMOVE auf das gerade ausgeführte Subkommando nur als letztes Kommando sinnvoll, da nachfolgende Kommandos des Subkommandos nicht mehr ausgeführt werden.
- Im Subkommando eines %CONTROL<sub>n</sub> dürfen Sie kein weiteres %CONTROL<sub>n</sub>-Kommando, keinen %INSERT, %JUMP (COBOL, FOR1) oder %ON angeben.



## Beispiele

1. `%INSERT S'20' <%DISPLAY A,B;%SET A INTO B;ADD-FILE-LINK...;%REM %•>`

Wenn der Programmablauf an die Anweisung mit der Nummer 20 kommt, gibt AID den Inhalt der Variablen A und B aus, weist der Variablen B den Wert von A zu und ruft das SDF-Kommando ADD-FILE-LINK auf. Da das Subkommando auch ein %REMOVE %• enthält, wird es nach seiner Ausführung gleich wieder gelöscht.

2. `%ON %LPOV <%DISPLAY %LINK>`

Immer wenn während des Programmablaufs ein Modul nachgeladen wird, gibt AID seinen Namen aus. Der Programmablauf wird fortgesetzt.

## 5.4 Kommandodateien

AID-Kommandos können auch in BS2000-Prozedurdateien und ENTER-Jobs stehen. Soll ein Eingabesatz mit einem AID-Kommando beginnen, so muss als erstes Zeichen ein Schrägstrich stehen, dem dann das %-Zeichen des AID-Kommandos folgt. Eine Marke für /SKIP-COMMANDS bzw. /GOTO muss jedoch noch vor dem %-Zeichen stehen.

Steht in einer BS2000-Prozedur ein AID-Kommando, das eine Quittung verlangt (z.B. /%AID CHECK=ALL.../%SET...), so setzt AID im Stapelbetrieb als Antwort Y ein.

### Beispiel

```
/LOAD-EXECUTABLE-PROGRAM *LIB-ELEM(TESTLIB,TESTLLM), TEST-OPT=AID
/ANF: %SET 17 INTO SLF
/%INSERT S'71' <%DISPLAY I,J,K>
.
.
.
/GOTO ANF
/END: EXIT-PROC
```

---

## 6 Subkommando

### 6.1 Beschreibung

Ein Subkommando ist ein Operand der Überwachungskommandos %CONTROL<sub>n</sub>, %INSERT und %ON. Mit diesen Kommandos legen Sie eine Überwachungsbedingung fest, die erfüllt sein muss, damit das zugehörige Subkommando bearbeitet wird. Sie haben also die Möglichkeit, steuernd in den Testablauf einzugreifen. Sie können automatisierte Testabläufe aufbauen, in denen an vorbestimmten Stellen im Programmablauf z.B. aktuelle Datenstände in Protokoll-Dateien ausgegeben oder Datenfeld- oder Registerinhalte modifiziert werden.

subkdo-OPERAND - - - - -

$$\langle [\text{subkdoname}:] [(\text{bedingung}):] \left[ \left. \begin{array}{l} \text{AID-kommando} \\ \text{BS2000-kommando} \end{array} \right\} \{; \dots\} \right] \rangle$$

- - - - -

Über den Subkommandonamen können Sie im weiteren Testverlauf das Subkommando ansprechen, um sich z.B. die Anzahl der Durchläufe des Subkommandos ausgegeben zu lassen oder um das Subkommando wieder zu löschen. Die Ausführung des Subkommandos kann von einer Bedingung abhängen, die zwischen dem Subkommandonamen und dem Kommandoteil stehen muss. Der Kommandoteil kann aus einem einzelnen Kommando oder aus einer Kommandofolge bestehen und AID- und BS2000-Kommandos enthalten (siehe [Abschnitt „Kommandofolge und Subkommandos“ auf Seite 47](#)).

Geben Sie in einem %INSERT, %CONTROL<sub>n</sub> oder %ON kein Subkommando an, setzt AID das Subkommando <%STOP> ein.

Wenn Sie jedoch *subkdoname* oder *bedingung* angeben und den Kommandoteil weglassen, ergänzt AID kein %STOP, sondern verhält sich am Testpunkt bzw. beim Eintreten des vereinbarten Ereignisses so, als ob ein %CONTINUE ergänzt würde:

- der Durchlaufzähler wird erhöht und kann gegebenenfalls über %\**subkdoname* abgefragt werden,
- das Programm läuft weiter,
- ein %TRACE wird fortgesetzt.

Adressoperanden im Kommandoteil eines Subkommandos, die keine vollständige explizite Qualifikation enthalten, werden bei der Eingabe entsprechend den aktuellen Vereinbarungen zur Basisqualifikation (siehe %BASE) bzw. zur *vorqualifikation* (siehe %QUALIFY) ergänzt.

Bei der Eingabe wird ein Subkommando nur syntaktisch geprüft. Ob die angegebenen symbolischen Adressen in den LSD-Sätzen verzeichnet sind oder ob zu einem Programmteil, der über eine Qualifikation angesprochen wird, überhaupt LSD-Sätze geladen sind, prüft AID erst bei der Ausführung des Subkommandos. Die entsprechenden LSD-Sätze müssen also bei der Eingabe des Subkommandos noch nicht geladen sein. Ebenso wird bei der Verwendung von Qualifikationen bei der Eingabe des Subkommandos noch nicht geprüft, ob die damit bezeichneten Programmteile existieren oder geladen sind.

Die Subkommandos der Kommandos %INSERT oder %ON können Sie ketten, und zwar nach dem LIFO-Prinzip. Sie können also entsprechend dem Testverlauf durch spätere Eingaben Subkommandos modifizieren bzw. aktualisieren. Dazu finden Sie im [Abschnitt „Ketten“ auf Seite 62](#) ausführliche Informationen. In den Subkommandos zu %INSERT und %ON können Sie weitere %INSERT- und %ON-Kommandos definieren. Dies ist im [Abschnitt „Schachteln“ auf Seite 65](#) beschrieben. Es gibt Kommandos, die in Subkommandos nicht zugelassen sind und/oder die zum Abbruch des Subkommandos, des Programms oder gar der Task führen. Alle Informationen darüber finden Sie im [Kapitel „Kommandoeingabe“ auf Seite 45](#). Dort steht auch, wie Fehler in Subkommandos behandelt werden.

## Beispiele

1. `%CONTROL1 %STMT IN (S'20':S'27') <%DISPLAY A_ARR>`

Jeweils vor der Ausführung der Anweisungen 20 bis 27 wird der Inhalt aller Elemente des Feldes A\_ARR ausgegeben.

2. `%INSERT INPUT <%DISPLAY INDAT;%SET KEY INTO I-KEY>`

Jedes Mal, wenn der Programmablauf den Paragraphen mit dem Namen INPUT erreicht, wird der Eingabesatz INDAT ausgegeben und der Inhalt des Schlüsselfeldes KEY nach I-KEY übertragen.

3. `%ON %LPOV <%SDUMP %NEST>`

Nach jedem Laden eines neuen Segmentes wird die aktuelle Aufrufhierarchie ausgegeben.

4. `%INSERT V'2C' <%SET #'2C' INTO %5; %DISPLAY 'INS_2C!!!'>`

Vor der Ausführung des Befehls mit der Adresse V'2C' lädt AID das Register 5 mit dem Wert #'2C' und gibt den Text "INS\_2C!!!" aus.

5. %ON %SVC <%C1 %INSTR <%R>; %C2 %INSTR <%REM %C>; %T 2 %INSTR>; %R

Alle SVCs ab der Eingabe des obigen %ON-Kommandos werden protokolliert. Zunächst wird mit %ON %SVC festgelegt, dass vor der Ausführung eines SVC das anschließende Subkommando ausgeführt wird. Das Subkommando enthält zwei %CONTROL-Kommandos und einen %TRACE, jeweils mit dem Kriterium %INSTR. Der %TRACE führt den nächsten Befehl aus, eben den SVC und protokolliert ihn. Die Ausführung des nächsten Befehls löst die Bearbeitung der beiden Subkommandos zu %CONTROL1 und %CONTROL2 aus: mit %RESUME wird der Programmablauf fortgesetzt, bis der nächste SVC erkannt wird; mit dem zweiten Subkommando (%REMOVE %CONTROL) wird der %CONTROL sofort wieder zurückgesetzt, da sonst zu jedem nachfolgenden Befehl ein %RESUME durchgeführt würde, was den Programmablauf stark verlangsamen würde.

## 6.2 Name und Durchlaufzähler

Der Subkommandoname besteht aus bis zu 30 Zeichen; das erste Zeichen kann A-Z, \$, @ oder Unterstrich "\_" sein. Die folgenden Zeichen können zusätzlich die Ziffern 0-9 und den Bindestrich "-" enthalten. Zu beachten ist, dass der Bindestrich am Ende einer Kommandozeile im Dialog stets als Fortsetzungszeichen interpretiert wird. Abgeschlossen wird der Subkommandoname mit einem Doppelpunkt, der jedoch nicht Teil des Namens ist.

Der Name muss nach der öffnenden spitzen Klammer stehen. Er muss eindeutig sein; gleiche Subkommandonamen weist AID mit einer Fehlermeldung ab. Bei geschachtelten Subkommandos prüft AID den Namen nicht bei der Eingabe, sondern erst, wenn das Subkommando ausgeführt wird. Insbesondere können innere Subkommandos aus einer Schachtelung, die mehrmals durchlaufen werden, nur dann einen Namen erhalten, wenn sie jedes Mal nach ihrer Ausführung explizit gelöscht werden.

Sie können bis zu 256 verschiedene Subkommandonamen vergeben.

Enthält das Subkommando ein %STOP, so wird der Subkommandoname in der STOP-Meldung mit ausgegeben.

Um den Subkommandonamen als allgemeinen AID-Operanden verwenden zu können, müssen Sie die Zeichen %• voran stellen. So entsteht ein AID-Schlüsselwort %•*subkdoname*, mit dem Sie im weiteren Testverlauf das Subkommando und den zugehörigen Durchlaufzähler ansprechen können.

Auf alle Subkommandos, auch wenn sie keinen Namen haben, können Sie innerhalb des Subkommandos mit der Kurzform %• Bezug nehmen. Außerhalb des Subkommandos können Sie Durchlaufzähler und Subkommando nicht mit der Kurzform %• ansprechen.

Der Durchlaufzähler hängt unmittelbar mit dem Subkommandonamen zusammen, da er über diesen angesprochen werden kann. Aber auch für Subkommandos, die keinen Namen haben, führt AID einen Durchlaufzähler, der dann nur innerhalb des zugehörigen Subkommandos mit der Kurzform %• abgefragt werden kann.

Der Durchlaufzähler ist ein numerischer Wert, der bei jeder Bearbeitung des Subkommandos um eins erhöht wird. Der Zähler wird auch dann erhöht, wenn das Subkommando eine Bedingung enthält und das Ergebnis der Bedingung FALSE ist und deshalb der zugehörige Kommandoteil nicht ausgeführt wird. Sie können selbst mit %MOVE oder %SET den Stand des Durchlaufzählers verändern. Der Durchlaufzähler kann auch einen negativen Wert annehmen.

Den aktuellen Stand des Durchlaufzählers erfahren Sie mit %DISPLAY %•*subkdoname* bzw. mit %DISPLAY %• für das gerade auszuführende Subkommando.

## Beispiele

1. %CONTROL1 %IO <IO: %CONTINUE>

Bei jeder Ein-/Ausgabeoperation des Programms wird der Durchlaufzähler %•IO um 1 erhöht.

2. %IN L'200' <L200: %DISPLAY %•IO; %STOP>

Das Programm hält an, wenn die Anweisungsmarke 200 erreicht ist. AID gibt den Stand des Durchlaufzählers aus, der zum Subkommando mit dem Namen %•IO aus Beispiel 1 gehört.

3. %CONTROL2 %CALL <PAR: %D %•,PAR1,PAR2,PAR3 P=MAX>

Das obige Kommando überwacht die Unterprogramm-Aufrufe. Dem Durchlaufzähler können Sie entnehmen, um die wievielte CALL-Anweisung es sich handelt. Zusätzlich zum jeweiligen Stand des Durchlaufzählers werden die Parameter PAR1, PAR2 und PAR3 nach SYSLST protokolliert (P=MAX).

Da das Subkommando den Namen PAR hat, können Sie sich jederzeit im weiteren Testverlauf mit %DISPLAY %•PAR ausgeben lassen, wie viele CALL-Aufrufe das Programm bis dahin durchlaufen hat.

## 6.3 Bedingte Ausführung

AID bietet Ihnen die Möglichkeit, die Ausführung eines Subkommandos von einer Bedingung abhängig zu machen. Die Bedingung muss in runde Klammern gesetzt und von einem Doppelpunkt abgeschlossen werden. Sie steht unmittelbar vor dem Kommandoteil des Subkommandos.

AID prüft die Bedingung und weist ihr den Wert TRUE oder FALSE zu. Nur bei TRUE wird der Kommandoteil ausgeführt, bei FALSE wird er übersprungen.

bedingung-OPERAND - - - - -

$$([\text{NOT}] \text{vergleich}_1 [ \left. \begin{array}{l} \text{AND} \\ \text{OR} \\ \text{XOR} \end{array} \right\} [\text{NOT}] \text{vergleich}_2 ] [ \dots ] ) :$$

Für *vergleich<sub>n</sub>* können Operanden gemäß der folgenden Syntax gebildet und eingesetzt werden:

vergleich-OPERAND - - - - -

$\left\{ \begin{array}{l} [\text{qua}] \left\{ \begin{array}{l} \text{datenname} \\ \text{anweisungsname} \\ \text{S}' \dots ' \\ \text{komp1-speicherref} \end{array} \right\} \\ \left\{ \begin{array}{l} \text{V}'f \dots f' \\ \text{C=csect} \\ \text{COM=common} \\ \text{schlüsselwort} \end{array} \right\} \\ \text{\%}(\dots) \\ \text{\%L}(\dots) \\ \text{\%L}(\text{ausdruck}) \\ \text{AID-literal} \end{array} \right\}$	$\left\{ \begin{array}{l} \text{EQ} \mid \text{NE} \\ \text{GE} \mid \text{LE} \\ \text{GT} \mid \text{LT} \\ \text{NG} \mid \text{NL} \end{array} \right\}$	$\left\{ \begin{array}{l} [\text{qua}] \left\{ \begin{array}{l} \text{datenname} \\ \text{anweisungsname} \\ \text{S}' \dots ' \\ \text{komp1-speicherref} \end{array} \right\} \\ \left\{ \begin{array}{l} \text{V}'f \dots f' \\ \text{C=csect} \\ \text{COM=common} \\ \text{schlüsselwort} \end{array} \right\} \\ \text{\%}(\dots) \\ \text{\%L}(\dots) \\ \text{\%L}(\text{ausdruck}) \\ \text{AID-literal} \end{array} \right\}$
---	--	---

Zusammenstellung der Vergleichs- und der Booleschen Operatoren:

Vergleichsoperatoren		Boolesche Operatoren	
EQ	gleich	NOT	logische Negation
NE	nicht gleich	AND	logisches UND
LE	kleiner oder gleich	OR	logisches ODER (inklusive)
LT	kleiner	XOR	logisches ODER (exklusiv)
NL	nicht kleiner		
GE	größer oder gleich		
GT	größer		
NG	nicht größer		

Die Vergleichsoperatoren sind untereinander gleichrangig und werden vor den Booleschen Operatoren abgearbeitet.

Bei den Booleschen Operatoren gilt die folgende Rangfolge:

NOT	höchster Rang
AND	zweithöchster Rang
OR/XOR	niedrigster Rang

Gleichrangige Operatoren werden von links nach rechts abgearbeitet. Um zu erreichen, dass die Operatoren in anderer Reihenfolge abgearbeitet werden, als durch die Rangfolge vorgegeben ist, müssen Sie entsprechende Klammern setzen.

### Beispiele

```
(I EQ J AND VAR EQ 'A' OR VAR EQ 'B'):
entspricht
(((I EQ J) AND (VAR EQ 'A')) OR (VAR EQ 'B')):

(NOT VAR EQ 'A' OR ADR NE %OG):
entspricht
((NOT (VAR EQ 'A')) OR (ADR NE %OG)):
```

Klammern sind auch dann zu setzen, wenn die Vergleichsoperatoren oder Booleschen Operatoren mit Variablenamen verwechselt werden können und dadurch eventuell als fehlerhafte Syntax abgewiesen werden.

Der einzige unäre Operator ist das NOT, d.h. er bezieht sich auf nur einen Operanden. Alle anderen Booleschen Operatoren und alle Vergleichsoperatoren sind binäre Operatoren, sie verknüpfen zwei Operanden miteinander.

Mit den Vergleichsoperatoren können stets nur genau zwei Operanden verglichen werden; eine Kettung ist hier also nicht möglich. Geben Sie z.B. als Bedingung

```
(A EQ B EQ C): an, so wird dies von AID schon bei der Eingabe mit der Meldung
AID0271 Syntaxfehler abgelehnt. Die Bedingung müsste vielmehr durch
(A EQ B AND B EQ C): dargestellt werden.
```

Als Operand von Vergleichsoperatoren ist jede Speicherreferenz zugelassen, die Sie bei AID bilden können (siehe [Abschnitt „Speicherreferenzen“ auf Seite 78](#)). Wenn Sie Dateielemente aus Ihrem Programm verwenden, werden diese den folgenden Speichertypen zugeordnet:

- Binärstring (≙ %X)
- Character (≙ %C)
- numerisch (≙ %A, %F, %P, %D).



Speicherinhalte vom Typ Character können bis maximal 1000 Bytes in einer Bedingung verglichen werden. Logische Variablen können Sie vergleichen, wenn Sie mit einer Typmodifikation einen anderen Speichertyp festlegen (z.B. (ALOG%X EQ X'FF'): ). Von den Schlüsselwörtern können die Durchlaufzähler der Subkommandos, die AID-Register, alle Programmregister sowie der Befehlszähler (Program Counter) %PC für Vergleiche herangezogen werden.

Die AID-Literale sind ebenfalls als Vergleichsoperanden zugelassen. Bei Character-Literalen (C'x...x') verwendet AID stets den Code des Eingabemediums, das heißt den Coded-Character-Set des Terminals oder der Prozedurdatei mit den AID-Kommandos. Wenn Sie Speicherinhalte vom Typ Character in ASCII vergleichen wollen, müssen Sie den Vergleichstext in ein Sedezimal-Literal übersetzen, so hat z.B. C'Hugo' in ASCII den sedezimalen Wert X'4875676F'.

Beim Schreiben einer Bedingung müssen Sie darauf achten, dass die Operanden-Typen verträglich sind. Welche Vergleiche zulässig sind und wie der Vergleich durchgeführt wird, ersehen Sie aus der Tabelle auf der nächsten Seite.

Die Zulässigkeit des Vergleichs wird erst geprüft, wenn das Überwachungsereignis eingetreten ist. Im Fehlerfall gibt AID eine entsprechende Meldung aus und setzt das Ergebnis des Vergleichs auf FALSE, d.h. der Kommandoteil des Subkommandos wird nicht ausgeführt.

AID unterscheidet zwischen Binär-, Character- und numerischem Vergleich. Die Art des Vergleichs leitet AID aus dem Typ der beteiligten Operanden ab. Die Operanden einer Bedingung werden von AID in einer bestimmten, von der jeweiligen Programmiersprache unabhängigen Systematik konvertiert und verglichen, da Sie ja in einem Vergleich Datenelemente aus Modulen verknüpfen können, die in unterschiedlichen Programmiersprachen geschrieben sind. Daraus ergibt sich, dass das Ergebnis eines AID-Vergleichs nicht unbedingt mit dem Ergebnis eines entsprechenden Vergleichs in einer bestimmten Programmiersprache übereinstimmen muss:

- Beim Character-Vergleich wird der kürzere Operand logisch mit Leerzeichen aufgefüllt, und AID vergleicht zwei gleichlange Operanden, während z.B. Fortran das Ergebnis des Vergleichs stets mit FALSE bewertet, wenn die beteiligten Operanden in der Länge nicht übereinstimmen.
- Beim Binärvergleich wird rechts mit X'00' aufgefüllt und dann genauso verfahren wie beim Character-Vergleich.
- Bei numerischen Vergleichen können unterschiedliche Ergebnisse dadurch zustande kommen, dass AID bei der Konvertierung der Operanden mit einer anderen Genauigkeit arbeitet wie die jeweilige Programmiersprache.
- COBOL ordnet die numerisch druckaufbereiteten Variablen den numerischen Operanden zu, für AID gehören diese Variablen zum Speichertyp Character.

Insbesondere müssen Sie diesen Sachverhalt in Betracht ziehen, wenn Sie Operanden aus verschiedenen Programmiersprachen vergleichen wollen.

Die Vergleiche, die mit den Vergleichsoperatoren gebildet werden, dienen als Operanden der Booleschen Operatoren. Logische Variablen, wie sie z.B. Fortran kennt, können nicht eingesetzt werden.

Mit den Booleschen Operatoren können Sie auch mehr als zwei Vergleiche miteinander verknüpfen. Die Obergrenze wird bestimmt durch die Komplexität der Vergleichsoperanden und die Größe des AID-internen Eingabepuffers.

Der folgenden Tabelle können Sie entnehmen, wie die verschiedenen Operanden-Typen miteinander verglichen werden, bzw. welche Vergleiche nicht zulässig sind:

Speichertyp 1.Operand	Speichertyp 2. Operand					
	%X X' f...f' B' b...b'	numerisch	%C C'x...x' U'x...x'	%UTF16/ NATIONAL	num. Literal	Pointer
%X X' f...f' B' b...b'	bin	bin	bin	bin	–	bin
numerisch	bin	num	num(1)	numchar UTF16-chr	num	–
%C C'x...x' U'x...x'	bin	num(1)	char	UTF16-conv UTF16-chr	num(1)	–
%UTF16/ NATIONAL	bin	numchar UTF16-chr	UTF16-conv UTF16-chr	UTF16-chr	numchar UTF16-chr	–
num. Literal	–	num	num(1)	numchar UTF16-chr	num	–
Pointer	bin	–	–	–	–	bin

**bin** Binär-Vergleich

Es wird Bit für Bit von links nach rechts verglichen. Der kürzere Operand wird rechts mit Nullen (B'0') aufgefüllt.

**char** Character-Vergleich

Es wird Byte für Byte von links nach rechts verglichen. Der kürzere Operand wird rechts mit Blanks (X'40') aufgefüllt.

**num** numerischer Vergleich

Der arithmetische Wert der beiden Operanden wird verglichen.

**numchar**

Beim Vergleich wird von dem ganzzahligen, numerischen Feld die abdruckbare Ziffernfolge in der UTF16-Codierung verwendet.

## UTF16-chr

Character-Vergleich in UTF16-Codierung

Es wird Byte für Byte von links nach rechts verglichen. Das Auffüllen und Kürzen wird jedoch mit UTF16-Zeichen durchgeführt (Leerzeichen in 2 Byte-Codierung).



Ordnungsrelationen, die bei Vergleichen in EBCDIC-Codierung verwendet werden, werden durch den byteweisen UTF16-Vergleich von AID nicht mehr unterstützt.

## UTF16-conv

Wenn ein Operand vom Typ %UTF16 ist und der zu vergleichende Operand vom Typ %C oder ein C-/U-Literal ist, erfolgt durch UTF16-conv die Konvertierung nach %UTF16. Anschließend kann der Vergleich wie bei UTF16-chr durchgeführt werden.

num<sup>(1)</sup> Enthält ein Operand vom Typ Character nur Ziffern und ist er höchstens 19 Stellen lang, so wird er numerisch verglichen, falls der zweite Operand vom Typ numerisch ist.

Alle übrigen Operanden vom Typ Character können mit numerischen Speichertypen oder numerischen Literalen nicht verglichen werden.

- kein Vergleich möglich

Der Vergleich wird mit einer Fehlermeldung abgewiesen. Das Ergebnis des Vergleichs wird auf FALSE gesetzt.

## Numerische Speichertypen:

%A, %Y (entspricht %AL2)	Ganzzahl ohne Vorzeichen
%F, %H (entspricht %FL2)	Ganzzahl mit Vorzeichen
%P	gepackte Zahl
%D	Gleitpunktzahl
%PC	Befehlszähler (Program Counter)
alle Register	
%*[subkdoname]	Durchlaufzähler

sowie alle symbolisch adressierten Datenelemente vom Typ numerisch.



Nicht alle Datenelemente, die in den einzelnen Programmiersprachen numerisch behandelt werden, haben auch für AID einen numerischen Speichertyp; siehe hierzu die einzelnen sprachspezifischen Handbücher (%SET-Tabelle).

Numerische Literale:

[{±}]n	Ganzzahl
#' f...f'	Sedezimalzahl
[{±}]n.m	Dezimalpunktzahl
[{±}]mantisseE[{±}]exponent	Gleitpunktzahl

Die Speichertypen %S bzw. %SX sind in Vergleichen nur bedingt sinnvoll und daher in obiger Tabelle nicht aufgeführt. %S wird wie %XL2 und %SX wie %XL4 behandelt.

Weitere Informationen zu Speichertypen und Literalen finden Sie in dem [Kapitel „AID-Literale“ auf Seite 113](#) und dem [Kapitel „Schlüsselwörter“ auf Seite 125](#).

## Beispiele

1. %IN S'18' <(%• LT 10): %D I,J; %MOVE X'58' INTO V'348'>

Wenn das Programm die Anweisung mit der Nummer 18 erreicht, unterbricht AID den Programmablauf und prüft die Bedingung des Subkommandos. Die ersten 9 Mal wird der Kommandoteil ausgeführt, d.h. AID gibt den Wert der Variablen I und J am Bildschirm aus und setzt den Inhalt der virtuellen Adresse V'348' auf X'58'. Da das Subkommando kein %STOP enthält, wird der Programmablauf bei der Anweisung mit der Nummer 18 fortgesetzt.

Bei jedem weiteren Durchlaufen des Testpunkts S'18' wird der Kommandoteil des Subkommandos übersprungen.

2. %IN S'25' <INS25: (ACHAR EQ 'END'): %D ISUM,JSUM; %STOP>

Das Subkommando des Testpunkts S'25' wird solange nicht ausgeführt, bis das Feld mit der symbolischen Adresse ACHAR den Inhalt 'END' hat. Dann zeigt AID den Inhalt der Summenfelder ISUM und JSUM am Bildschirm an, und das Programm wechselt in den Kommandomodus.

3. %CONTROL1 %IO <OUTPUT: (SLF NE 200): %D %•, OUTDAT, SLF P=MAX>

Beim Testen eines Programmes, das Sätze mit der Satzlänge 200 ausgeben sollte, stellen Sie fest, dass die Satzlänge nicht immer stimmt. Mit dem obigen Kommando %CONTROL1 können Sie das Satzlängenfeld SLF überwachen. Jedes Mal wenn ein Satz ausgegeben wird, aber SLF nicht den Wert 200 enthält, protokolliert AID den Inhalt des Durchlaufzählers sowie der Variablen OUTDAT und SLF nach SYSLST.

4. %INSERT S'18' <IN1:(I EQ 15): %SET 1 INTO J; %D ARRAY(K),K;%STOP>

Auf die Anweisung mit der Nummer 18 wird ein Testpunkt gesetzt. Zu diesem Testpunkt wird ein Subkommando mit dem Namen IN1 eingetragen: Immer dann, wenn der Index I den Wert 15 hat, wird Index J auf 1 gesetzt und das Vektorelement ARRAY(K) mit dem zugehörigen Index K ausgegeben. Danach hält das Programm an.

5. %SET 0 INTO %0G  
%INSERT S'25' <ZL1: (CNO NE '3'): %SET %L=(1 + %0G) INTO %0G;-  
%D %, %0G, CNO, OUTDAT P=MAX>

Der %SET setzt das AID-Register %0G auf 0. Mit dem %INSERT wird auf die Anweisung mit der Nummer 25 ein Testpunkt gesetzt und das Subkommando mit dem Namen ZL1 definiert. Bei jedem Durchlaufen des Testpunkts erhöht AID den Zähler %•ZL1 um 1. Nur wenn die Kennziffer CNO am Testpunkt nicht auf '3' steht, wird auch Register %0G um 1 erhöht, und AID protokolliert die Zählerstände, den Inhalt der Kennziffer CNO und den Ausgabesatz OUTDAT nach SYSLST (P=MAX).

Der Inhalt von %•ZL1 gibt also an, wie oft das Programm Anweisung 25 durchläuft, während in %0G gezählt wird, wie oft die Kennziffer CNO am Testpunkt nicht auf 3 steht.

6. %IN PROC <(%3 GT 4096 AND %5->%L1 EQ 'A'): %SET %L=(%5 + 2) INTO %5>

Vor der Ausführung der Anweisung mit dem Namen PROC prüft AID, ob Register %3 einen Wert > 4096 enthält und ob gleichzeitig an der Speicherstelle, auf die Register %5 zeigt, ein 'A' steht. Trifft dies zu, wird der Inhalt von Register %5 um 2 erhöht und der Programmablauf fortgesetzt.

## 6.4 Ketten

Wenn Sie mehrere %INSERT für denselben *testpunkt* oder %ON für dasselbe *ereignis* schreiben, kettet AID das zuletzt eingegebene Subkommando vor das vorige (LIFO-Prinzip). Eine Ausnahme bildet das *write-ereignis* beim %ON. Hier kann nicht gekettet werden, sondern das zuletzt eingegebene Kommando überschreibt das vorherige. AID macht Sie mit der Warnung AID0496 darauf aufmerksam.

Gekettet wird auch, wenn das neuere Kommando kein explizites Subkommando hat. In diesem Fall wird dann das implizit erzeugte Subkommando <%STOP> vor das bereits eingetragene gesetzt, und dieses ältere Subkommando wird nie mehr ausgeführt. Sinnvoller ist allerdings, das nicht mehr benötigte Subkommando zuerst zu löschen, da AID sonst im weiteren Testverlauf einen Eintrag verwalten muss, der nie mehr bearbeitet wird.

Enthält ein gekettetes Subkommando eine Bedingung, so gilt diese Bedingung nur für den dazugehörigen Kommandoteil. Für die Behandlung der Subkommandos, die in der Kette nachfolgen, gibt es zwei Möglichkeiten:

- Der von der Bedingung abhängige Kommandoteil ist mit %CONTINUE, %RESUME, %TRACE oder %STOP abgeschlossen. Dann werden nachfolgende Subkommandos nur bearbeitet, wenn das Ergebnis der Bedingung FALSE ist.
- Der von der Bedingung abhängige Kommandoteil enthält kein %CONTINUE, %RESUME, %TRACE oder %STOP: Nachfolgende Subkommandos werden stets abgearbeitet unabhängig davon, ob die Bedingung zutrifft oder nicht.

Zu einem %CONTROL<sub>*n*</sub> können Sie dagegen keine Subkommandos ketten. Ein neu eingegebener %CONTROL<sub>*n*</sub> überschreibt alle Operandenwerte eines früheren %CONTROL<sub>*n*</sub> zu derselben Nummer *n* mit den Angaben aus dem neuen Kommando.

## Beispiele

1. Im Testverlauf geben Sie folgende Kommandos ein:

```
%ON %LPOV(SUBTOT)
.
.
.
%ON %LPOV(SUBTOT) <%DISPLAY S=B1@.PROC=B1.CHAR_DAT>
```

Beim ersten %ON ergänzt AID als Subkommando <%STOP>, da explizit kein Subkommando angegeben wurde. Durch Kettung entsteht nach der Eingabe des zweiten %ON für das Ereignis %LPOV(SUBTOT), d.h. nach dem Laden des Moduls SUBTOT das folgende Subkommando:

```
<%DISPLAY S=B1@.PROC=B1.CHAR_DAT; %STOP>
```

2. Wie sich die Kettung in Verbindung mit dem standardmäßig eingesetzten <%STOP> auswirkt, soll nun gezeigt werden:

```
%INSERT ST4 <%D TEXTDAT>
.
.
.
%INSERT ST4
```

Der zweite %INSERT enthält kein Subkommando. Dafür setzt AID ein <%STOP> ein. Da das zweite %INSERT-Kommando denselben Testpunkt bezeichnet wie das vorhergehende, kommt es zur Kettung, bei der folgendes Subkommando entsteht:

```
<%STOP; %DISPLAY TEXTDAT>
```

Die Ausführung eines Subkommandos wird aber durch %STOP abgebrochen. %DISPLAY TEXTDAT wird also nie mehr ausgeführt, bleibt jedoch als Subkommando zum Testpunkt ST4 eingetragen und kann aus der Kette auch nicht herausgelöscht werden, da es keinen Namen hat. Am besten geben Sie jedem Subkommando einen Namen, so dass Ihnen bei falscher Kettung stets die Möglichkeit bleibt, das Subkommando über seinen Namen aus der Kette zu löschen.

Richtiger wäre im obigen Beispiel gewesen, wenn Sie zunächst mit

```
%REMOVE ST4
```

den ersten %INSERT gelöscht und dann

```
%INSERT ST4
```

geschrieben hätten.

3. Die beiden folgenden %INSERTs können Sie in einer Prozedur einsetzen, um ein beliebiges Character-Literal zu suchen. Im Trefferfall steht die gefundene Adresse im AID-Register %0G und die Länge des gesuchten Strings in %2G (siehe %FIND).

```
%INSERT V'1648' <(%0G NE -1): %SET %L=(%1G - %0G) INTO %2G>
%INSERT V'1648' <%FIND C'x...x'>
```

Die Kettung wird notwendig, da Sie eine Bedingung nur am Anfang des Subkommandos schreiben können. Erst durch Kettung entsteht zum Testpunkt V'1648' das gewünschte Subkommando:

```
<%FIND C'x...x'; (%0G NE -1): %SET %L=(%1G - %0G) INTO %2G>
```

4. %INSERT S'50' <%D NO,INDAT; %STOP>  
 %INSERT S'50' <(ISW EQ X'FF'): %SET X'00' INTO ISW; %CONT>

Durch die beiden %INSERTs zur selben Source-Referenz, nämlich der Anweisung mit der Nummer 50, entsteht am Testpunkt S'50' das folgende bedingte Subkommando mit THEN- und ELSE-Zweig (zur Verdeutlichung sind die verschiedenen Teile der Konstruktion mit IF, THEN und ELSE gekennzeichnet):

```
(ISW EQ X'FF'): %S X'00' INTO ISW; %CONT; %D NO, INDAT; %STOP
┌ IF              ┌ THEN              ┌ ELSE
```

Jeweils vor der Ausführung der Anweisung S'50' unterbricht AID den Programmablauf und prüft den Inhalt von Schalter ISW. Enthält der Schalter den Wert X'FF', wird er auf X'00' zurückgesetzt und der Programmablauf fortgesetzt. Andernfalls gibt AID den Inhalt von NO und INDAT aus und hält das Programm an.



## 6.5 Schachteln

Im Subkommando eines %INSERT oder %ON kann wieder ein %INSERT oder %ON stehen. Das wird als Schachtelung von Subkommandos bezeichnet. AID unterstützt diese Schachtelung über mehrere Subkommando-Stufen. Wie tief Sie Subkommandos schachteln können, hängt von deren Komplexität und von der Größe des AID-internen Eingabepuffers ab.

Geschachtelte Subkommandos werden schrittweise wirksam. Während die Überwachungsbedingung der ersten Generation (äußere Schachtel) sofort von AID eingetragen wird und damit im nachfolgenden Programmablauf auch schon eine Unterbrechung erzeugen kann, trägt AID den *testpunkt* (%INSERT) oder das *write-ereignis* bzw. *ereignis* (%ON) in jüngeren Subkommandos erst dann ein, wenn die Überwachungsbedingung der unmittelbar vorhergehenden Generation zur Unterbrechung geführt hat. Tritt in einer Schachtelung eines %INSERT- oder eines %ON-Kommandos für eine schon eingetragene Überwachungsbedingung ein weiteres anderes Kommando auf, so kommt es zusätzlich zur Kettung des neuen Subkommandos vor das ältere (LIFO-Prinzip). Nicht gekettet wird dagegen ein Subkommando zu einem Testpunkt oder einem Ereignis einer inneren Schachtel, wenn der Testpunkt der nächsthöheren Stufe der Schachtelung auf Grund einer Programmschleife mehrmals durchlaufen wird bzw. wenn das Ereignis der äußeren Schachtel mehrmals eintritt.

Subkommandos zu einem %CONTROL<sub>n</sub> können Sie nicht schachteln, daher sind im Subkommando eines %CONTROL<sub>n</sub> zusätzlich zu den Kommandos, die in allen Subkommandos nicht erlaubt sind, auch die Kommandos %CONTROL<sub>n</sub>, %INSERT und %ON nicht zugelassen. In Subkommandos eines %CONTROL<sub>n</sub> darf außerdem kein %JUMP (COBOL, FOR1) angegeben werden.

## Beispiele

1. `%IN ST3 <%DISPLAY 'INSERT1', TEXTDAT;%IN OUTPUT <%D 'INSERT2', I,J,K,-  
NUM-TAB; %ON %SVC(186) <%D 'OPEN DAT1',I,J>>>`

Das Beispiel bezieht sich auf ein COBOL-Programm. Mit `%INSERT ST3` wird als Testpunkt der Paragraph `ST3` vereinbart. Dieser `%INSERT` enthält einen darin geschachtelten `%INSERT`, der wiederum ein `%ON`-Kommando enthält. Der darin angegebene Testpunkt `OUTPUT` sowie das angegebene Ereignis `%SVC(186)` ( $\hat{=}$ `OPEN`) haben noch keine Auswirkung auf den Ablauf. Sie können erst aktiv werden, wenn der Testpunkt des `%INSERT` erreicht wird, in dessen Subkommando sie definiert sind. Kommt der Programmablauf zur symbolischen Adresse `ST3`, wird das zugehörige *subkdo* ausgeführt, d.h. das Literal `'INSERT1'` und der Inhalt des Ausgabesatzes `TEXTDAT` werden ausgegeben und der Testpunkt `OUTPUT` wird gesetzt. Das Subkommando zum Testpunkt `OUTPUT` ist noch nicht wirksam. Im zu testenden Programm sind also bis zu dieser Stelle des Programmablaufs die Testpunkte `ST3` und `OUTPUT` gesetzt.

Da das Subkommando zum Testpunkt `ST3` kein `%STOP`-Kommando enthält, wird das Programm fortgesetzt. Erreicht der Programmablauf die Adresse `OUTPUT`, wird nun der `%DISPLAY 'INSERT2', I,J,K, NUM-TAB` ausgeführt. Neben diesem Kommando enthält das Subkommando noch einen `%ON` für das Ereignis `%SVC(186)`. Erkennt AID im weiteren Programmablauf einen `SVC` zum Öffnen einer Datei, führt es das Subkommando aus, das mit dem `%ON` vereinbart wurde: das Literal `'OPEN DAT1'` und der Inhalt der Indizes `I` und `J` werden ausgegeben.

2. `%IN ST4 <%D TEXTDAT>  
%ON %LPOV (SUBTOT) <%REMOVE ST4; %IN ST4 <%D 'SUBTOT LOADED'; %STOP >>  
%RESUME`

Immer wenn der Testpunkt `ST4` erreicht wird, gibt AID den Speicherinhalt des Datenfeldes `TEXTDAT` aus. Tritt das vereinbarte Ereignis `%LPOV (SUBTOT)` ein, d.h. der Modul `SUBTOT` wird geladen, führt AID das Subkommando im `%ON`-Kommando aus. Der Testpunkt `ST4` wird gelöscht, aber es wird sofort wieder zu diesem Testpunkt ein neues Subkommando eingetragen:

```
<%DISPLAY 'SUBTOT LOADED'; %STOP>
```

Wenn `ST4` das nächste Mal durchlaufen wird, gibt AID den Text `'SUBTOT LOADED'` aus und unterbricht den Programmablauf. Sie können wieder neue Kommandos eingeben.

## 6.6 Löschen

Zum Löschen von Subkommandos steht das Kommando %REMOVE zur Verfügung. Implizit löschen Sie ein Subkommando, indem Sie das zugehörige Überwachungskommando oder bei %INSERT den zugehörigen Testpunkt oder bei %ON das zugehörige Ereignis löschen.

Explizit können Sie ein Subkommando über seinen Namen löschen. Diese Möglichkeit besteht nur für Subkommandos eines %CONTROL $n$  oder eines %INSERT. Da zu einem %CONTROL $n$  Subkommandos nicht gekettet werden können, bewirkt ein %REMOVE %*subkdoname* dasselbe wie %REMOVE %CONTROL $n$ . Mit %INSERT können jedoch zu einem bestimmten Testpunkt eine ganze Reihe von Subkommandos hintereinander gekettet werden. In diesem Fall löscht %REMOVE %*subkdoname* ein einzelnes Subkommando über seinen Namen aus der Kette heraus. Wenn das Subkommando keinen Namen hat, kann es nur mit dem gesamten Testpunkt gelöscht werden. Es empfiehlt sich daher, Subkommandos stets einen Namen zu geben.

Bei geschachtelten Subkommandos können innere Subkommandos, die noch nicht am zugehörigen Testpunkt eingetragen wurden, nicht aus der Schachtelung herausgelöscht werden, auch nicht über ihren Namen. Diese Subkommandos können Sie nur zusammen mit dem gesamten %INSERT (%REMOVE %INSERT) oder mit dem Testpunkt (%REMOVE *testpunkt*) löschen.

Das aktuelle Subkommando können Sie unmittelbar nach seiner Ausführung wieder löschen, indem Sie als letztes Kommando des Kommandoteils %REMOVE %*•* schreiben. Der %REMOVE %*•* wird sofort ausgeführt, was zur Folge hat, dass eventuell noch nachfolgende Kommandos ebenfalls gelöscht werden und somit nicht mehr ausgeführt werden können.



---

## 7 Adressierung in AID

Die Kommandos zur Ablaufüberwachung, das Kommando %JUMP (Festlegen einer Fortsetzungsadresse) und die Kommandos zur Ausgabe und Modifikation von Speicherinhalten verlangen Operanden, die eine Adresse oder einen Bereich im Speicher bezeichnen. Eine Adresse im ausführbaren Teil des Programms müssen Sie bei %DISASSEMBLE, %INSERT, %JUMP und %REMOVE angeben. %CONTROL<sub>n</sub> und %TRACE verlangen als Operanden einen Speicherbereich im ausführbaren Teil des Programms, während bei den Kommandos %DISPLAY, %FIND, %MOVE und %SET der anzugebende Speicherbereich auch im Datenteil des Programms liegen kann.

Eine Adresse bezeichnen Sie bei AID durch eine Adresskonstante oder durch eine komplexe Speicherreferenz. Einen Speicherbereich können Sie abhängig vom Kommando durch eine Qualifikation oder eine Speicherreferenz angeben oder Sie beschreiben den Bereich durch zwei Adressen, der Bereich liegt dann zwischen der ersten und der zweiten Adresse. Ausführlich beschrieben finden Sie die anzugebenden Operanden bei den Kommandobeschreibungen in den sprachspezifischen Handbüchern und im Handbuch für das Testen auf Maschinencode-Ebene.

Das Kommando %SDUMP nimmt eine Sonderstellung ein, der zugehörige Operand *dump-bereich* bezeichnet entweder einen Namensraum, der mit einer Qualifikation angegeben werden kann, oder ein einzelnes Datum.

In den folgenden Abschnitten finden Sie alle Begriffe, mit denen Sie bei AID eine Adresse bezeichnen können, die verschiedenen Qualifikationen, die einfachen und die komplexen Speicherreferenzen, beschrieben.

## 7.1 Qualifikationen

Mit Qualifikationen beschreiben Sie den Pfad zu einem Speicherobjekt, das außerhalb des gerade gültigen AID-Arbeitsbereiches oder nicht im aktuellen Haupt- oder Unterprogramm liegt oder dort nicht eindeutig ist. In einigen Fällen kann die Adressierung mit einer Qualifikation enden, Sie sprechen dann mit der Qualifikation das Speicherobjekt selbst an. Es gibt die Basisqualifikation und Bereichsqualifikationen. Qualifikationen geben Sie immer in der Reihenfolge der Übergeordneten zur Untergeordneten an und nur soweit sie zur eindeutigen Pfadbeschreibung nötig sind. Überflüssige Qualifikationen ignoriert AID.

Aufeinanderfolgende Qualifikationen werden durch einen Punkt getrennt. Außerdem muss zwischen der letzten Qualifikation und dem anschließenden Adressteil ein Punkt stehen. Mit %QUALIFY können Sie Qualifikationen vordefinieren. In einem Adressoperanden können Sie die mit %QUALIFY vereinbarten Qualifikationen durch Voran Stellen eines Punktes übernehmen.

### 7.1.1 Basisqualifikation

Die Basisqualifikation bezeichnet die Umgebung (Environment), sie legt fest, ob eine anschließende Adresse im virtuellen Speicher oder in einer Dump-Datei liegen soll. Die Basisqualifikation wird beim symbolischen und maschinennahen Testen gleich verwendet.

AID unterstützt eine Basisqualifikation (E=...) nicht innerhalb eines Operanden, sondern nur am Anfang. Bei Funktionen wie Adressfunktion %@(...) oder Längenfunktion %L(), %L=(), ... muss eine Basisqualifikation vor diesen Funktionen angegeben werden.

E=VM ist der Standardwert und bezeichnet den virtuellen Speicherbereich des geladenen Programms.

E=Dn bezeichnet einen Speicherabzug in einer Dump-Datei mit einem Linknamen D0 - D7; die Dump-Datei muss zuvor mit %DUMPFILe zugewiesen werden.

Die Basisqualifikation können Sie global mit %BASE vereinbaren oder im Adressoperanden für eine einzelne Speicherreferenz angeben. Als alleiniger Operand ist die Basisqualifikation in den Kommandos %BASE, %QUALIFY und %SDUMP zugelassen. In allen übrigen Adress- oder Bereichsoperanden müssen Sie an eine Basisqualifikation einen der folgenden Begriffe anschließen:

- Bereichsqualifikation
- Datename
- Anweisungsname
- Source-Referenz
- virtuelle Adresse
- Schlüsselwort

## 7.1.2 Bereichsqualifikationen

Mit einer Bereichsqualifikation bezeichnen Sie einen Teilbereich eines Programms. Beim Programmieren, Übersetzen oder Binden eines Programms legen Sie die Teilbereiche fest und/oder benennen sie. Bereichsqualifikationen geben Sie an, wenn eine Adresse nicht in dem Programmteil liegt, der gerade durchlaufen wird. Es gibt unterschiedliche Bereichsqualifikationen für das maschinennahe Testen und für die jeweilige Programmiersprache. Die Bereichsqualifikationen für das symbolische Testen werden von der Struktur der Programmiersprache bestimmt. Im [Kapitel „Voraussetzungen zum Testen mit AID“ auf Seite 35](#) der sprachspezifischen Handbücher ist beschrieben, welche Programmteile durch welche Qualifikationen angesprochen werden.

SPID=X'f...f'	maschinennah (AR-Modus)
ALET={X'f...f'  %nAR %nG}	maschinennah (AR-Modus)
CTX=kontext	symbolisch und maschinennah
L=ladeeinheit	maschinennah
O=objektmodul	maschinennah
C=csect/segmentname/sharename	maschinennah/COBOL
COM=common	symbolisch und maschinennah
S=srcname	alle Programmiersprachen
PROC=name	alle Programmiersprachen
PROG=name	Assembler, COBOL, Fortran
ONUNIT='onunitname'	PL/I
BLK=' blkname'	C++/C, PL/I
NESTLEV=level-nummer	alle Programmiersprachen

Die Bereichsqualifikationen geben Sie im Adressoperanden zu einer Speicherreferenz an. Dort werden sie zur Pfadbeschreibung eingesetzt. Sie müssen nur die Qualifikationen angeben, die zur eindeutigen Ansprache benötigt werden. Liegt die Unterbrechungsstelle jedoch in den Routinen des Laufzeitsystems, können Sie Daten und Anweisungen im eigenen Programm nur mit der vollen Qualifikation ansprechen.

In den Kommandos, die einen Bereichsoperanden erfordern, können Sie alle Bereichsqualifikationen außer SPID und ALET einsetzen, um damit einen Bereich zu bezeichnen.

C=csect und COM=common können Sie zusätzlich auch als Anfangsadresse einsetzen. Alle Bereichsqualifikationen können Sie mit %QUALIFY als Vorqualifikation vereinbaren.

In einer komplexen Speicherreferenz müssen Sie darauf achten, dass Sie mit den nachfolgenden Operationen die Bereichsgrenzen nicht überschreiten. Obwohl Sie auf das Attribut Länge einer Bereichsqualifikation nicht zugreifen können, wird dennoch überprüft, ob das Ergebnis eines Adressversatzes oder einer Längenmodifikation noch innerhalb des Bereichs liegt, der mit der Qualifikation angegeben wurde.

Die **ALET- und SPID-Qualifikation** bezeichnen einen Datenraum und können nur vor einer virtuellen Adresse oder einer komplexen Speicherreferenz, die ohne symbolische Komponenten gebildet wird, verwendet werden.

Die **Kontextqualifikation** bezeichnet den Kontext, in dem die mit nachfolgenden Qualifikationen oder Adressangaben angesprochenen Speicherbereiche oder Adressen liegen sollen. Sie ist nur erforderlich, wenn eine CSECT, ein COMMON oder eine Übersetzungseinheit in mehreren Kontexten enthalten ist und die aktuelle Unterbrechungsstelle nicht in der CSECT, dem COMMON oder der Übersetzungseinheit liegt, in dem das mit dem Adressoperanden ausgewählte Speicherobjekt enthalten ist.

Angegeben wird die Kontextqualifikation mit `CTX=kontext`. Dabei ist *kontext* ist der explizit im BIND-Makro mit dem Operand `LNKCTX[@]` vergebene Name oder der implizite Name `LOCAL#DEFAULT`, falls `LNKCTX[@]` nicht angegeben wurde. Dynamisch mit dem DBL geladene Programme erhalten den gleichen standardmäßig vergebenen Kontextnamen `LOCAL#DEFAULT`. Weitere Kontexte eines Programms können durch Konnektieren an ein Shared-Code-Programm (z.B. an ein DSSM-Subsystem oder an ein Programm in einem COMMON-MEMORY-POOL) entstehen.

Die Operanden *vorqualifikation* des Kommandos `%QUALIFY` und *dump-bereich* im `%SDUMP` können mit `CTX=kontext` enden. In allen übrigen Adress- oder Bereichsoperanden müssen Sie an eine CTX-Qualifikation einen der folgenden Begriffe anschließen:

- eine weitere Bereichsqualifikation
- Datenname
- Anweisungsname
- Source-Referenz

Die **L- und O-Qualifikation** geben Sie an, wenn Sie den Pfad zu einer von mehreren gleichnamigen CSECTs oder COMMONs beschreiben müssen. Sie müssen nur die L- und/oder O-Qualifikation angeben, die zur eindeutigen Ansprache genügt. Auf L- und/oder O-Qualifikation muss stets eine C- oder COM-Qualifikation folgen.

Die **C- und COM-Qualifikation** können Sie in den Kommandos `%CONTROLn` und `%TRACE` als Bereichsangabe verwenden. Wenn Sie mit der C-Qualifikation eine CSECT bzw. mit der COM-Qualifikation einen COMMON bezeichnen, können Sie nur ein maschinennahes Kriterium angeben. Die Angabe `C=sharename/segmentname`, die Sie beim Testen von COBOL-Programmen einsetzen können, darf nur mit einem symbolischen Kriterium kombiniert werden. Auf eine C-Qualifikation kann jedoch nie eine symbolische Speicherreferenz folgen, auch nicht in COBOL.

In den Kommandos `%DISASSEMBLE`, `%INSERT` und `%REMOVE` wird mit der C-/COM-Qualifikation die Anfangsadresse der CSECT/des COMMON angegeben. Auch in den Kommandos `%DISPLAY`, `%FIND`, `%MOVE`, `%ON %WRITE(...)` und `%SET` kann der Adressoperand auf `C=csect/COM=common` enden. Es wird damit die gesamte CSECT bzw. der gesamte COMMON angesprochen. Die CSECT bzw. der COMMON wird in diesen Kommandos als maschinennahe Speicherreferenz verwendet. Ebenso können Sie innerhalb einer komplexen Speicherreferenz die C-/COM-Qualifikation als Speicherreferenz einsetzen (siehe [Abschnitt „Maschinennahe Speicherreferenzen“ auf Seite 80](#)).



Die **S-, PROC-, BLK-, ONUNIT- und PROG-Qualifikation** können Sie zur Bezeichnung eines Speicherbereichs in %CONTROL<sub>n</sub> und %TRACE oder des Namensraumes bei %SDUMP angeben. Sie bezeichnen damit den gesamten angegebenen Programmteil. Als Speicherreferenz können Sie diese Qualifikationen nicht verwenden.

Auf eine S-Qualifikation kann folgen:

- eine PROC-, BLK- oder ONUNIT-Qualifikation
- ein Datename
- ein Anweisungsname
- eine Source-Referenz

Auf eine PROC-, BLK- oder ONUNIT-Qualifikation kann folgen:

- ein Datename
- ein Anweisungsname
- eine Source-Referenz

Die **PROG-Qualifikation** ist eine Zusammenfassung für S=srcname•PROC=name, falls *srcname* und *name* gleich sind. Sie kann in Assembler, COBOL und Fortran eingesetzt werden.

Die **NESTLEV-Qualifikation** können Sie in den Kommandos %DISPLAY, %MOVE, %SDUMP and %SET verwenden. In diesen Kommandos bezeichnet die Qualifikation NESTLEV=*level-nummer* die Ebene in der Aufrufhierarchie, die mit dem AID-Kommando %SDUMP %NEST erzeugt wird. Ein Beispiel zur Verwendung der NESTLEV-Qualifikation finden Sie auf [Seite 74](#).

## Beispiele

1. %BASE E=VM  
%DUMPFIL E D1=M.DUMP  
%DISPLAY V'10A', E=D1.V'10A'

Mit %BASE vereinbaren Sie den virtuellen Speicherbereich des geladenen Programms als Basisqualifikation. Mit %DUMPFIL E weisen Sie dem Linknamen D1 die Datei M.DUMP zu.

Da die Basisqualifikation E=VM gilt, gibt AID für die erste Angabe im %DISPLAY-Kommando vier Bytes ab Adresse V'10A' des geladenen Programms aus und für die zweite Angabe vier Bytes ab Adresse V'10A' aus einem Speicherabzug aus. Dieser Speicherausgang steht in einer Datei, die mit %DUMPFIL E dem Linknamen D1 zugewiesen wurde.

4 Bytes ist die implizite Länge einer V-Adresse.

2. %DISPLAY S=COMPUTE@.PROC=COMPUTE.SUM

Aus der Programmeinheit COMPUTE@ eines Fortran-Programms gibt AID den Inhalt der Variablen SUM aus. S- und PROC-Qualifikation sind nötig, wenn das Programm in einer anderen Programmeinheit unterbrochen wurde.

3. `%QUALIFY E=D1.S=COMPUTE@.PROC=COMPUTE`  
`%DISPLAY .SUM`  
 Vor den Punkt vor SUM setzt AID die vereinbarte Vorqualifikation. So entsteht das Kommando: `%DISPLAY E=D1.S=COMPUTE@.PROC=COMPUTE.SUM`  
 AID gibt aus der Dump-Datei, die dem Linknamen D1 zugewiesen ist, aus der Programmeneinheit `COMPUTE@` das Datenfeld SUM aus.
4. `%DISPLAY E=D2.S=TEST@.BLK='23'.var`  
 In der Dump-Datei mit dem Linknamen D2, die den Speicherabzug eines C-Programms enthält, und dort in der Übersetzungseinheit mit dem Codemodulnamen `TEST@` liegt im Block, der in Zeile 23 beginnt, die lokale Variable var. Deren Inhalt gibt AID aus.
5. `%MOVE L=LAD1.C=CS1.(%L(L=LAD1.C=CS1) - 4) INTO %2G`  
 AID überträgt die letzten 4 Bytes der CSECT CS1 aus der Ladeeinheit LAD1 in das AID-Register %2G. Die L-Qualifikation ist nötig, da CS1 nicht die aktuelle CSECT ist und der Name CS1 nicht eindeutig innerhalb des Programmsystems ist.
6. `%INSERT S=COMPUTE@.PROC=COMPUTE.S'16' <%DISPLAY %.>`  
 AID setzt einen Testpunkt auf die Anweisung mit der Nummer 16 in der Programmeneinheit `COMPUTE@`. Kommt der Programmablauf an diesen Testpunkt, wird der Durchlaufzähler ausgegeben und das Programm fortgesetzt.
7. Beispiel für die Verwendung der NESTLEV-Qualifikation

Folgendes Programm soll getestet werden:

```
#include <stdio.h>

int main(void) {
    int arr[6] = {0, 0, 0, 0, 0, 0};
    int i = 0;
    proca(i, arr);
}

int proca(int i, int arr[])
{
    int testVar;
    testVar = 10 + i;
    arr[i] = i;
stop: ;
    if (i < 4)
        procb(i + 1, arr);
    arr[5] = arr[5] + testVar;
    return 0;
}

int procb(int i, int arr[])
{
    int testVar;
```

```

    testVar = 20 + i;
    arr[i] = i;
    if (i < 4)
        { proca(i + 1, arr); }
    arr[5] = arr[5] + testVar;
    return 0;
}

```

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

```

%AID C=YES
%INSERT stop
%RESUME
%RESUME

```

Die aktuelle Aufrufhierarchie:

```

/MSD %NEST
SRC_REF: 15 SOURCE: CREC PROC: proca *****
SRC_REF: 27 SOURCE: CREC BLK : 27 *****
SRC_REF: 28 SOURCE: CREC PROC: procb *****
SRC_REF: 16 SOURCE: CREC PROC: proca *****
SRC_REF: 6 SOURCE: CREC PROC: main *****
ABSOLUT: V'1019108' SOURCE: ICS$MAI@ PROC: ICS$MAI@ *****
ABSOLUT: V'10020D8' SOURCE: IC@MAIN@ PROC: IC@MAIN@ *****

```

Die aktuelle Aufrufhierarchie mit Aufrufebenen:

Die Zahl in Spalte 3 ist die einfache Aufruftiefe (NESTLEV). Die Angabe bei RLEV : ist die Rekursionstiefe. Sie hat nur informativen Character und kann nicht zur Steuerung von AID genutzt werden.

```

/%AID LEV=ON
/MSD %NEST
# 1 SRC_REF: 15 SOURCE: CREC RLEV: 0 PROC: proca *****
# 2 SRC_REF: 27 SOURCE: CREC RLEV: 0 BLK : 27 *****
# 3 SRC_REF: 28 SOURCE: CREC RLEV: 0 PROC: procb *****
# 4 SRC_REF: 16 SOURCE: CREC RLEV: 1 PROC: proca *****
# 5 SRC_REF: 6 SOURCE: CREC RLEV: 0 PROC: main *****
# 6 ABSOLUT: V'1019108' SOURCE: ICS$MAI@ PROC: ICS$MAI@ *****
# 7 ABSOLUT: V'10020D8' SOURCE: IC@MAIN@ PROC: IC@MAIN@ *****

```

Die Daten des rekursiven `proca` auf den verschiedenen Ebenen:

```

/%SD NESTLEV=1
# 1 SRC_REF: 15 SOURCE: CREC RLEV: 0 PROC: proca *****
stop           = 010001CA
i              =                2
arr           = 010E181C
testVar       =                12

```

```

/%SD NESTLEV=4
# 4 SRC_REF: 16 SOURCE: CREC RLEV: 1 PROC: proca *****
stop           = 010001CA
i              =                0
arr           = 010E181C
testVar       =                10

```

NESTLEV kombiniert mit der Basisqualifikation `E=VM`:

```

/%D E=VM.NESTLEV=4.testVar
testVar       =                10

```

Um eine Variable auf einer bestimmten Aufrufebene zu verändern, verwenden Sie `%SET` oder `%MOVE`:

```

/%MOVE X'00000064' INTO NESTLEV=4.testVar
/%D NESTLEV=4.testVar %X
V'010E18C8' = testVar + #'00000000'
010E18C8 (00000000) 00000064          ...

```

NESTLEV kann für den Sender und für den Empfänger verwendet werden:

```

/%SD testVar
# 1 SRC_REF: 15 SOURCE: CREC RLEV: 0 PROC: proca *****
testVar       =                12
# 3 SRC_REF: 28 SOURCE: CREC RLEV: 0 PROC: procb *****
testVar       =                21
# 4 SRC_REF: 16 SOURCE: CREC RLEV: 1 PROC: proca *****
testVar       =                100

/%SET NESTLEV=4.testVar INTO NESTLEV=3.testVar
/%D NESTLEV=3.testVar
testVar       =                100

```

Nach dem Ausführen von %RESUME sieht die aktuelle Aufrufhierarchie folgendermaßen aus:

```

/%R
/%SD %NEST
# 1 SRC_REF: 15 SOURCE: CREC RLEV: 0 PROC: proca *****
# 2 SRC_REF: 27 SOURCE: CREC RLEV: 0 BLK : 27 *****
# 3 SRC_REF: 28 SOURCE: CREC RLEV: 0 PROC: procb *****
# 4 SRC_REF: 16 SOURCE: CREC RLEV: 1 PROC: proca *****
# 5 SRC_REF: 27 SOURCE: CREC RLEV: 1 BLK : 27 *****
# 6 SRC_REF: 28 SOURCE: CREC RLEV: 1 PROC: procb *****
# 7 SRC_REF: 16 SOURCE: CREC RLEV: 2 PROC: proca *****
# 8 SRC_REF: 6 SOURCE: CREC RLEV: 0 PROC: main *****
# 9 ABSOLUT: V'1019108' SOURCE: ICS$MAI@ PROC: ICS$MAI@ *****
#10 ABSOLUT: V'10020D8' SOURCE: IC@MAIN@ PROC: IC@MAIN@ *****

```

Die rekursive Funktion proca zeigt die folgenden Instanzen von testVar:

```

/%SD PROC='proca'.testVar
# 1 SRC_REF: 15 SOURCE: CREC RLEV: 0 PROC: proca *****
testVar = 14
# 4 SRC_REF: 16 SOURCE: CREC RLEV: 1 PROC: proca *****
testVar = 12
# 7 SRC_REF: 16 SOURCE: CREC RLEV: 2 PROC: proca *****
testVar = 100

```

## 7.2 Speicherreferenzen

Mit einer Speicherreferenz können Sie in einem AID-Kommando ein Speicherobjekt ansprechen. Beschreibt die Speicherreferenz eine Zeichenfolge, z.B. vom Typ %C oder %UTF16, können die Zeichenkonvertierungsfunktionen %C() oder %UTF16() auf die Speicherreferenz angewendet werden.

AID unterscheidet zwischen einfachen und komplexen Speicherreferenzen.

Beispiele für einfache Speicherreferenzen sind:

- virtuelle Adressen: V'f...f'
- Datennamen: VAR1, FELD(I)
- Schlüsselwörter: %14, %2D, %PC, %CLASS6
- C-Qualifikationen: C=CS1
- COM-Qualifikationen: COM=CB
- Anweisungsnamen: L'20', COMPUTE1
- Source-Referenzen: S'133', S'44ADD'

Beispiele für komplexe Speicherreferenzen sind:

- %@(VAR1)->.(%L=(I+5))%XL20
- C=CS1.#'100'%SX->%CL8

Die einfachen Speicherreferenzen (das sind die maschinennahen und die symbolischen Speicherreferenzen und die Schlüsselwörter), sind mit ihren Attributen und Eigenschaften im [Abschnitt „Maschinennahe Speicherreferenzen“ auf Seite 80](#) beschrieben.

Eine komplexe Speicherreferenz ist eine Vorschrift, nach der AID eine Adresse errechnet bzw. nach der Sie die Attribute eines Speicherobjekts modifizieren können. In einer komplexen Speicherreferenz können Sie symbolische und maschinennahe Speicherreferenzen, Schlüsselwörter, Konstanten und AID-Literale in den Operationen Adressversatz, indirekte Adressierung, Typ- und Längenmodifikation und Adressselektion verwenden, um die in einer Testsituation benötigte Adresse von AID errechnen zu lassen bzw. um Typ und Länge einer Speicherreferenz festzulegen. Informationen zur komplexen Speicherreferenz und den zugehörigen Operationen stehen im [Abschnitt „Symbolische Speicherreferenzen“ auf Seite 82](#).

## Attribute

Attribute beschreiben die Eigenschaften eines Speicherobjekts oder einer Konstanten. Speicherobjekte haben bis zu sechs Attribute:

- Name (optional)
- Adresse
- Inhalt
- Länge
- Speichertyp
- Ausgabebetyp

Auf die Attribute Adresse, Länge und Speichertyp können Sie mit Selektoren zugreifen. Mit der Modifikation können Sie Länge und Speichertyp verändern. Das Attribut Länge bestimmt zugleich die Bereichsgrenzen: Adresse bis Adresse + (Länge - 1). Die Bereichsgrenzen werden bei den Operationen Längenmodifikation und Adressversatz überprüft. Bei der Übertragung mit %MOVE wird überprüft, ob *sender* innerhalb der Bereichsgrenzen von *empfänger* Platz hat. Eine Ausnahme bildet die virtuelle Adresse, der als Bereich der gesamte Benutzer-Adressraum zugeordnet ist, obwohl das Längenattribut nur 4 Bytes beträgt.

Wie AID im jeweiligen Kommando die Attribute berücksichtigt und welchen Prüfungen sie unterzogen werden, ist bei den einzelnen Kommandos in den sprachspezifischen Handbüchern bzw. in dem Handbuch für das Testen auf Maschinencode-Ebene beschrieben. Wie AID innerhalb der Berechnung einer komplexen Speicherreferenz diese Attribute berücksichtigt, ist bei den Operationen der komplexen Speicherreferenz beschrieben.

Konstanten haben kein Adressattribut. Sie können deshalb nur in bestimmten Zusammenhängen verwendet werden. Insbesondere kann die Adressselektion nicht auf sie angewendet werden.

## 7.2.1 Maschinennahe Speicherreferenzen

Maschinennahe Speicherreferenzen sind die CSECTs, die COMMONs und die virtuellen Adressen.

Die **CSECTs** und **COMMONs** werden in Form der C- bzw. COM-Qualifikation mit  $C=csect$  bzw.  $COM=common$  angegeben. Da die CSECTs/COMMONs wie Qualifikationen angegeben werden und in einigen Kommandos auch wie Qualifikationen verwendet werden können, sind sie auch im [Abschnitt „Bereichsqualifikationen“ auf Seite 71](#) beschrieben.

Als Speicherreferenz besitzt die C-/COM-Qualifikation die folgenden Attribute:

Name
Adresse
Inhalt
Länge (Länge der CSECT/des COMMON)
Speichertyp (%X)
Ausgabety (Dump)

Die Bereichsgrenzen werden durch die Anfangsadresse und die Länge der CSECT/des COMMON festgelegt.

Die folgenden Operationen können auf eine C-/COM-Qualifikation angewandt werden:

- Adressselektor
- Längenselektor
- Adressversatz
- Typmodifikation
- Längenmodifikation

Eine **virtuelle Adresse** geben Sie in folgender Form an:

$V'f...f'$ , wobei  $f...f'$  eine maximal 8stellige Sedezimalzahl zwischen '0' und '7FFFFFFF' ist. Mit einer virtuellen Adresse sprechen Sie direkt eine Speicherstelle im geladenen Programm oder einer Dump-Datei an. Im AR-Modus kann damit auch eine Speicherstelle in einem Datenraum angesprochen werden. Hierzu ist die Angabe einer ALET-/SPID-Qualifikation vor der virtuellen Adresse nötig. Ansonsten ist vor einer virtuellen Adresse nur die Angabe einer Basisqualifikation sinnvoll.

Das Ergebnis eines Adressversatzes oder einer indirekten Adressierung ist ebenfalls eine virtuelle Adresse und hat dann auch deren Attribute.



Attribute einer virtuellen Adresse sind:

Adresse (f...f)  
 Inhalt  
 Länge (4 Bytes)  
 Speichertyp (%X)  
 Ausgabetyt (Dump)

Die Bereichsgrenzen reichen von V'0' bis V'7FFFFFFF'.

Im Gegensatz zu allen anderen Speicherobjekten, bei denen die Adresse und die Länge zugleich die Bereichsgrenzen festlegt, gelten bei der virtuellen Adresse andere Bereichsgrenzen. Der gesamte Benutzer-Adressraum ist für Operationen mit virtuellen Adressen möglich und nur die niedrigste Adresse V'0' und die höchste mögliche Adresse V'7FFFFFFF' dürfen nicht überschritten werden.

Auf eine virtuelle Adresse kann folgen:

- Adressversatz (•)
- indirekte Adressierung (->)
- Typmodifikation
- Längenmodifikation

## Beispiele

1. %DISPLAY V'100'->>->%C

Die 4 Bytes ab Adresse V'100' haben den Inhalt X'00000A1A'. Adresse V'A1A' hat den Inhalt X'0000000F' (erster Pointer-Operator). Adresse V'F' hat den Inhalt X'0000B001' (zweiter Pointer-Operator). Adresse V'B001' hat den Inhalt X'F1F2F3F4' (dritter Pointer-Operator). Den interpretiert AID als Character und gibt '1234' aus.

2. %MOVE E=D1.V'206'.(%1)->.(%2-5) INTO %2G

In der Dump-Datei D1 soll ab Adresse V'206' um den Inhalt in Register %1 (X'00000004') weiterpositioniert werden. Der dort (V'20A') stehende Speicherinhalt (X'0000B111') wird als Adresse für eine Pointer-Operation verwendet. Ab dieser neuen Adresse (V'B111') wird um Inhalt von Register %2 (X'00000008') minus 5 weiterpositioniert und 4 Bytes ab dieser Adresse (V'B114') in das AID-Register %2G übertragen.

## 7.2.2 Symbolische Speicherreferenzen

Mit symbolischen Speicherreferenzen werden die symbolischen Adressen bezeichnet, die der Compiler bei der Übersetzung in den LSD-Sätzen hinterlegt. Dazu gehören die von Ihnen im Programm vergebenen Namen von Daten und Anweisungen, also Marken, Entrys oder Funktionsnamen sowie die vom Compiler erzeugten Source-Referenzen, mit denen Sie jede ausführbare Anweisung Ihres Programms ansprechen können, unabhängig davon, ob die Anweisung eine Marke hat oder nicht. Wenn also LSD-Sätze erzeugt wurden und zur Verfügung stehen, kann AID über Daten- oder Anweisungsnamen oder über Source-Referenzen auf die zugehörigen Adressen und die mit der Adresse verbundenen Attribute zugreifen.

Anweisungsnamen und Source-Referenzen sind Adresskonstanten und werden erst mit einem nachfolgenden Pointer-Operator zur Speicherreferenz. Ohne Pointer-Operator können sie nur in den Kommandos verwendet werden, die eine Adresse als Operanden verlangen. Wenn Sie jedoch den Befehlscode ansprechen wollen, der an der entsprechenden Adresse im Speicher steht, müssen Sie den Pointer-Operator anschließen.

### 7.2.2.1 Datennamen

Datennamen sind Namen von Variablen, Datenstrukturen, Feldern, Matrizen oder Vektoren, je nachdem was die jeweilige Programmiersprache anbietet und wie sie es benennt. In Tabellen oder Strukturen können Sie mit AID auf die einzelnen Elemente wie in einer Anweisung der Programmiersprache zugreifen. Sie setzen also erforderliche Kennzeichnungen, Indizes oder Subskripte hinter den Datennamen. Die wenigen Abweichungen sind in den Kommandobeschreibungen der sprachspezifischen Handbücher angegeben.

Im Quellprogramm definierte Konstanten gehören ebenfalls zu den Datennamen. Sie werden z.B. mit EQU (Assembler), *literal* und *symbolic character* im SPECIAL NAMES-Paragrafen (COBOL) oder mit PARAMETER (Fortran) vereinbart. Da sie keinen Speicherplatz belegen, können sie jedoch nicht wie alle anderen Daten verwendet werden. Sie haben kein Adressattribut. AID steht nur der Wert der Konstanten zur Verfügung. Die übrigen Attribute können Sie nicht verwenden.

Die Attribute von Datennamen werden im Quellprogramm festgelegt. Nur den Ausgabetyt legt AID selbst analog der Speichertyp-Ausgabetyt-Zuordnung fest (siehe [Abschnitt „Allgemeine Speichertypen“ auf Seite 125](#)). Datennamen haben die Attribute:

Name
Adresse
Inhalt
Länge
Speichertyp
Ausgabetyt

Die Bereichsgrenzen werden durch Adresse und Länge bestimmt.

Datenamen können Sie in allen Kommandos, die den Datenteil adressieren, angeben. Mit Selektoren können Sie auf die Attribute Adresse, Länge und Speichertyp zugreifen, um ihre Ergebnisse ausgeben oder übertragen zu lassen, um sie zu modifizieren oder um auf die Maschinencode-Ebene zu wechseln.

Auf einen Datennamen kann angewandt werden oder folgen:

- Adressselektor
- Längenselektor
- Typselektor
- Zeichenkonvertierungsfunktion
- Adressversatz (•)
- Längenmodifikation
- Typmodifikation
- indirekte Adressierung (->), aber nur wenn der Datename vom Typ %A ist.

Mit einer Typmodifikation können Sie den Speichertyp oder den damit verbundenen Ausgabetyt verändern. Für die Kommandos %DISPLAY und %SET müssen der Speicherinhalt und der mit der Modifikation vereinbarte Speichertyp zusammen passen.

Mit der Längenmodifikation können Sie von der mit einem Datennamen verbundenen Länge abweichen. Der Datentyp bleibt nicht erhalten, AID nimmt den Speichertyp %X an. Mit der Längenmodifikation dürfen Sie die Bereichsgrenzen nicht überschreiten, d.h. die modifizierte Länge darf nicht größer als die implizite Länge aus dem Längenattribut sein.

Wenn Sie an die impliziten Attribute eines Datennamens nicht mehr gebunden sein wollen, wenden Sie den Adressselektor auf den Datennamen an und setzen dahinter einen Pointer-Operator. Mit %@(datename)-> sprechen Sie dann die virtuelle Adresse eines Datennamens an, und es gelten die Attribute einer virtuellen Adresse.

### Indizes und Subskripte

Ist ein Datename der Name einer tabellarischen Struktur, kann er wie in einer Anweisung der Programmiersprache indiziert werden. COBOL unterscheidet zwischen Indizierung und Subskribierung, wobei die Subskribierung der Indizierung bei anderen Programmiersprachen entspricht. Besonderheiten bei der Behandlung des COBOL-Indizes durch AID sind im Handbuch „Testen von COBOL-Programmen“ beschrieben.

Der Index kann folgendermaßen angegeben werden:

$$\left. \begin{array}{l} n \\ \text{datename} \\ \text{arithmetischer ausdrück} \end{array} \right\}$$

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

**datenname**

bezeichnet den zu dem Vektor definierten Index oder eine numerische Variable, die im selben Programmteil wie der Vektor liegen muss; d.h. für den Index wird die Qualifikation des Vektors übernommen.

**arithmetischer Ausdruck**

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

Für COBOL gilt, dass in einem arithmetischen Ausdruck nur das Subskript, nicht aber der Index verwendet werden kann.

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

*index1:index2*

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

**Beispiele**

## 1. %DISPLAY V'10A'%T(SYMBOL)

Der Speicherinhalt ab Adresse V'10A' wird mit dem Speichertyp von SYMBOL interpretiert und in dem zugehörigen Ausgabetyt und der Länge von SYMBOL ausgegeben. AID prüft, ob der Speicherinhalt von V'10A' und der Speichertyp von SYMBOL vereinbar sind.

## 2. %DISPLAY %@(DATAREC)-&gt;.4%T(INPUT)

Für DATAREC gibt es keine Datenstruktur-Beschreibung; aber die Struktur von INPUT entspricht der von DATAREC mit dem Unterschied, dass in DATAREC eine 4 Bytes lange Nummer voransteht. Mit Adressselektor und nachfolgendem Pointer-Operator wird die virtuelle Adresse von DATAREC verwendet. Damit haben die Bereichsgrenzen keine Bedeutung mehr. Mit dem Adressversatz werden die ersten 4 Bytes von DATAREC übersprungen. Mit dem Typselektor wird der Speichertyp und die Länge von INPUT ausgewählt und der Speicherinhalt ab der errechneten Adresse entsprechend ausgegeben.

### 7.2.2.2 Anweisungsnamen und Source-Referenzen

Anweisungsnamen sind die Namen, die im Quellprogramm für Marken, Kapitel, Paragraphen oder Marken-/Eingangskonstanten bzw. -variablen vergeben wurden, je nachdem was die jeweilige Programmiersprache anbietet und wie sie es benennt. Die Anweisungsnamen stehen für die Adresse des zu der ersten Anweisung nach der Marke generierten Befehlscodes.

Anweisungsnamen geben Sie in folgender Form an:

L'nummer'	Marke (Fortran)
L'name'	Marke (alle Programmiersprachen); in %DISASSMBLE, %INSERT, %REMOVE ohne L' ...' möglich, falls keine Adressrechnung anschließt.
name	Funktion (C++/C), Programmname (COBOL, Fortran, Assembler; kann nur in den Kommandos %DISASSEMBLE, %INSERT und %REMOVE zur Bezeichnung des Programmanfangs verwendet werden), Entrykonstante oder -variable (PL/I)

Source-Referenzen sind die vom Compiler erzeugten Nummern oder Namen von Anweisungen, die in den LSD-Sätzen hinterlegt sind und mit denen Sie die Anweisungen ansprechen können, die weder am Anfang eines Haupt- oder Unterprogramms stehen noch eine Marke haben. Wie der jeweilige Compiler die Anweisungen benennt und was er dazu in der Übersetzungsliste eingetragen, ist in den sprachspezifischen Handbüchern beschrieben. Die Source-Referenzen stehen für die Anfangsadresse des zu einer Anweisung generierten Befehlscodes.

Source-Referenzen geben Sie in folgender Form an:

S'nummer/name'

#### Eigenschaften

Source-Referenzen und Anweisungsnamen sind Adresskonstanten. Sie belegen keinen Speicherplatz, haben also kein Adressattribut und können somit auch nicht verändert werden. Eine Ausnahme bilden Entry- und Markenvariablen bei PL/I, für die wie für alle anderen Daten Speicherplatz angelegt wird und die Sie mit %SET überschreiben können.

Wenn LSD-Sätze erzeugt wurden und zur Verfügung stehen, kann AID über Anweisungsnamen und Source-Referenzen auf den Befehlscode zugreifen.

Als einfache Speicherreferenzen können Sie Anweisungsnamen und Source-Referenzen in den Kommandos %DISASSEMBLE, %JUMP, %INSERT und %REMOVE verwenden. Mit Source-Referenzen können Sie außerdem in den Kommandos %CONTROL<sub>n</sub> und

%TRACE einen Speicherbereich angeben. Bei %DISPLAY, %MOVE und %SET sprechen Sie den Wert der Adresskonstanten an, den Sie allerdings nur als *sender* verwenden können.

Eine Ausnahme bilden Kapitel- und Paragraphennamen bei COBOL. AID kennt von ihnen nicht nur die Adresse des ersten Befehls, sondern auch das Ende des Kapitels oder Paragraphen. Sie können sie deshalb zusätzlich bei den Kommandos %CONTROL<sub>n</sub> und %TRACE zur Bereichsangabe verwenden.

Ansonsten können Sie mit Anweisungsnamen und Source-Referenzen nur dann eine Speicherstelle ansprechen, wenn ihnen ein Pointer-Operator folgt. AID steht nur die Adresskonstante zur Verfügung. Die übrigen Attribute können Sie nicht verwenden.

Auf einen Anweisungsnamen kann nur die indirekte Adressierung (->) folgen.

## Beispiele

1. %DISPLAY L'SUMMIEREN'

%DISPLAY L'SUMMIEREN'->

Mit dem ersten %DISPLAY gibt AID die Adresse des Befehlscodes aus, der zu der ersten Anweisung nach der Marke SUMMIEREN generiert wurde.

Mit dem zweiten %DISPLAY gibt AID 4 Bytes Speicherinhalt ab dieser Adresse aus.

2. %INSERT S'123'

%INSERT S'123'->.(-6)

Im ersten %INSERT setzen Sie einen Testpunkt auf die Adresse des Befehlscodes, der zu der Anweisung 123 generiert wurde. Sie benutzen die Source-Referenz S'123' als einfache Speicherreferenz.

Im zweiten %INSERT setzen Sie einen Testpunkt auf die Adresse des Befehlscodes, der 6 Bytes vor dem Testpunkt aus dem ersten %INSERT liegt. Sie benutzen die Source-Referenz S'123' in einer komplexen Speicherreferenz und müssen deshalb ihre Eigenschaften als Adresskonstante beachten und den Pointer-Operator dahinter schreiben.

3. %MOVE L'123' INTO %2G

%MOVE X'D2' INTO L'123'->

Zu der Marke 123 in einem Fortran-Programm sei die Adresse V'A1A' in den LSD-Sätzen hinterlegt. Mit dem ersten %MOVE überträgt AID diese Adresse in das AID-Register %2G. Mit dem zweiten %MOVE überträgt AID das Sedezimal-Literal X'D2' auf die Speicherstelle mit der Adresse V'A1A'.

### 7.2.3 Schlüsselwörter

Speicherobjekte, die außerhalb des Programmspeichers liegen, aber vom Programm oder von AID benutzt werden, kann AID über Schlüsselwörter ansprechen. Das sind die Mehrzweckregister %0-%15, die Gleitpunktregister %nE, %nD und %nQ, die Zugriffsregister %nAR, der Befehlszähler %PC, die AID-Register %0G-%15G und %nGD sowie der Durchlaufzähler %*subkdoname*. Außerdem können Sie den Klasse-5- und Klasse-6-Speicher mit den Schlüsselwörtern %CLASS5, -ABOVE und -BELOW und %CLASS6, -ABOVE und -BELOW ansprechen. Alle anderen Schlüsselwörter können nicht als Speicherreferenz verwendet werden. Vor einem Schlüsselwort kann nur eine Basisqualifikation angegeben werden.

Alle Schlüsselwörter, die Sie bei AID verwenden können, sind im [Kapitel „Schlüsselwörter“ auf Seite 125](#) beschrieben.

Schlüsselwörter haben die Attribute:

Name (%name)
Adresse
Inhalt
Länge
Speichertyp
Ausgabotyp

Die Bereichsgrenzen werden durch die Anfangsadresse und die Länge bestimmt.

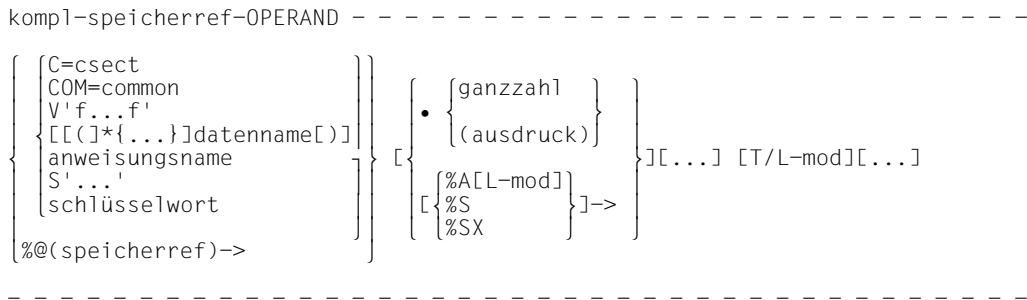
Auf ein Schlüsselwort kann angewandt werden oder folgen:

- Adressselektor (das Ergebnis ist nicht benutzbar, wenn die Adresse außerhalb des Benutzerbereichs liegt)
- Längenselektor
- Adressversatz (•)
- indirekte Adressierung (->)
- Längenmodifikation
- Typmodifikation

Bei Adressversatz und Längenmodifikation überprüft AID die Bereichsgrenzen. Mehrzweckregister können Sie ohne Typmodifikation vor einem Pointer-Operator verwenden, obwohl sie vom Typ %F sind.

### 7.2.4 Komplexe Speicherreferenzen

In einer komplexen Speicherreferenz wird, von einer symbolischen oder maschinennahen Speicherreferenz oder von einem Schlüsselwort ausgehend, eine Adressrechnung durchgeführt. Das Ergebnis einer komplexen Speicherreferenz ist ohne abschließende Typ- und Längenmodifikation eine virtuelle Adresse mit Speichertyp %XL4. Mit einer Typ- oder einer Längenmodifikation oder beiden Modifikationen können Sie der errechneten Adresse jedoch den von Ihnen gewünschten Speichertyp zuordnen.



C=csect	C-Qualifikation
COM=common	Common-Qualifikation
V'f...f'	virtuelle Adresse
datenname	Datenname
anweisungsname	Anweisungsname
S'...'	Source-Referenz
schlüsselwort	Schlüsselwort
%@(speicherref)	Adresselektor
T/L-mod	Typ- und/oder Längenmodifikation
%A, %S, %SX	Speichertypen zur Adressinterpretation
• {...}	Adressversatz (Byte-Offset)
->	indirekte Adressierung (Pointer-Operator)
*	indirekte Adressierung (Inhaltsoperator, nur C++/C)
(ausdruck)	arithmetischer Ausdruck

Adressversatz, indirekte Adressierung, Typ- und Längenmodifikation, arithmetischer Ausdruck und der Adresselektor sind in den unmittelbar folgenden Abschnitten beschrieben. Alle übrigen Begriffe finden Sie am Anfang dieses Kapitels beschrieben.



### 7.2.4.1 Adressversatz "."

Mit dem Adressversatz oder Byte-Offset können Sie in Byte-Schritten von einer Adresse aus vorwärts oder zurück gehen. Das Ergebnis eines Adressversatzes ist immer eine virtuelle Adresse. Beim Adressversatz dürfen die Bereichsgrenzen des betroffenen Speicherobjekts nicht überschritten werden.

adressversatz - - - - -

$$\text{speicherreferenz} \cdot \left\{ \begin{array}{l} \text{zahl} \\ \text{(ausdruck)} \end{array} \right\}$$

- - - - -

- Offset-Operator

speicherreferenz

kann jede, wie auch immer angesprochene Speicherstelle sein: virtuelle Adresse, Datename, Schlüsselwort, C-Qualifikation oder komplexe Speicherreferenz.

zahl

ist eine positive ganze Dezimal- oder Sedezimalzahl zwischen 0 und  $2^{31}-1$ .

ausdruck

ist ein von AID errechneter Wert zwischen  $-2^{31}$  und  $2^{31}-1$ . *ausdruck* ist in [Abschnitt „Adress-, Typ- und Längenselektor“ auf Seite 102](#) beschrieben. Er wird gebildet aus Zahlen, numerischen Inhalten von Speicherreferenzen, dem Ergebnis von Adress- und Längenselektor sowie Längenfunktion und den arithmetischen Operatoren (+ - \* /).

Der Adressversatz kann nur innerhalb der Bereichsgrenzen des betroffenen Speicherobjekts durchgeführt werden. Das Ergebnis eines Adressversatzes ist eine virtuelle Adresse mit der Länge 4. Diese 4 Bytes müssen innerhalb der Bereichsgrenzen des Speicherobjekts Platz haben. Werden die Bereichsgrenzen überschritten, gibt AID eine Fehlermeldung aus.

Außer bei den virtuellen Adressen werden die Bereichsgrenzen durch die Anfangsadresse und das Längenattribut bestimmt. Für eine virtuelle Adresse ist der gesamte virtuelle Speicherbereich (V'0' bis V'7FFFFFFF') als Bereich zugeordnet. Bei Datennamen, den Schlüsselwörtern %CLASS6, -ABOVE, -BELOW und der C-Qualifikation können Sie mit Adressselektion und nachfolgendem Pointer-Operator, %@(...)->, die symbolische Ebene verlassen, dann gelten die Bereichsgrenzen einer virtuellen Adresse.

Auf Adressversatz kann folgen:

- Adressversatz (•)
- indirekte Adressierung (->)
- Längenmodifikation
- Typmodifikation

## Beispiele

1. `%DISPLAY SYMBOL.10`  
`%DISPLAY %@(SYMBOL)->.10`  
 SYMBOL hat die Länge 10. Ein Adressversatz um 10 Bytes kann AID nicht durchführen, da dadurch die ersten 4 Bytes nach SYMBOL angesprochen und somit die Bereichsgrenzen von SYMBOL überschritten würden. AID gibt eine Fehlermeldung aus. Wenn Sie mit Adressselektion und nachfolgendem Pointer-Operator auf die Maschinencode-Ebene wechseln, gelten die Bereichsgrenzen einer virtuellen Adresse. Mit dem Adressversatz positionieren Sie wie im ersten %DISPLAY auf das erste Byte nach SYMBOL. Diesen %DISPLAY kann AID jetzt ausführen und gibt die ersten 4 Bytes nach SYMBOL aus.
2. `%D %@(VAR)->.(%L(ELEM(1))*5)%T(ELEM(1))`  
 In einem COBOL-Programm sei ein Vektor ELEM mit 10 Elementen ELEM(1) bis ELEM(10) enthalten. Die Variable VAR soll als Vektor in der Struktur von ELEM redefiniert werden, und das 6. Element davon soll ausgegeben werden. AID geht ab der Anfangsadresse von VAR mit einem Adressversatz um den Wert vorwärts, der sich aus der Länge von ELEM multipliziert mit 5 ergibt. Durch die anschließende Typmodifikation erreichen Sie, dass der Inhalt an der errechneten Adresse in Typ und Länge eines Elementes von ELEM ausgegeben wird.  
 Hätten Sie ELEM ohne Index angegeben, dann hätte AID Typ und Länge des gesamten Vektors ELEM genommen.
3. `%D %5->.(%L(INDEX)*%L(ADRESSE))%CL=(%L(ADRESSE)+50)`  
 Der Inhalt von Register 5 (X'0000A00') wird als Adresse eingesetzt. Ab Adresse V'A00' wird um die Länge von Index (2) multipliziert mit der Länge von ADRESSE (7) weiterpositioniert. Der Speicherinhalt an der Adresse V'A0E' ( $\#A00' + (2*7) = \#A0E'$ ) wird im Character-Format in der Länge 57 (7+50) ausgegeben.
4. `%D S'123COMP'->.8%S->%L10`  
 Für die Source-Referenz S'123COMP' (COBOL) ist die Adresse V'1B0' in den LSD-Sätzen hinterlegt. Mit dem Pointer-Operator wird auf die Stelle im Speicher mit der Adresse V'1B0' positioniert. Mit dem Adressversatz wird um 8 Bytes weitergegangen. Der Inhalt X'600F0130' an dieser Stelle, das ist an Adresse V'1B8', wird mit %S interpretiert. Aus Basisregister 6, Inhalt X'000B010', und Distanz #'00F' ergibt sich die Adresse V'B01F', die mit dem Pointer-Operator angesprochen wird. Ab dieser Adresse werden 10 Bytes im Dump-Format ausgegeben.
5. `%D C=CS1.(%L(C=CS1))`  
`%D C=CS1.(%L(C=CS1)-4)`  
 Der erste %DISPLAY wird abgewiesen, da ein Adressversatz in der Länge von CS1 das erste Byte hinter CS1 anspricht und somit die Bereichsgrenzen von CS1 überschritten würden.  
 Beim zweiten %DISPLAY ist der Adressversatz um 4 Bytes kürzer. Die Bereichsgrenzen von CS1 werden nicht verletzt und AID gibt die letzten 4 Bytes von CS1 aus.

6. %D V'4' .(-5) .4  
%D V'4' .(4-5)  
%D V'4' .4.(-5)

Der Adressversatz aus dem ersten %DISPLAY wird abgelehnt. Mit (-5) wird die untere Bereichsgrenze von virtuellen Adressen V'0' verletzt, auch wenn das Endergebnis durch den zweiten Adressversatz innerhalb des zulässigen Bereichs liegen würde. Im zweiten und dritten %DISPLAY werden bei jedem Adressversatz die Bereichsgrenzen eingehalten. AID gibt 4 Bytes ab Adresse V'3' aus.

7. %D V'100' .(%1 + %2)

Die Adresse V'100' wird um die Summe aus dem Inhalt von Register 1 und Register 2 erhöht.

#### 7.2.4.2 Indirekte Adressierung "->" / "\*"

Bei einer indirekten Adressierung benutzt AID eine Adresskonstante oder einen Speicherinhalt als Adresse für eine andere Speicherstelle. Wird der Pointer-Operator als unärer Operator eingesetzt, dann ist das Ergebnis eine virtuelle Adresse. Die Pointer-Operation bewirkt also einen Übergang auf die Maschinencode-Ebene. Wird die Indirektion mit dem Pointer-Operator als binärem Operator durchgeführt oder verwenden Sie den Inhaltsoperator, dann bleiben Sie auch nach der indirekten Adressierung auf der symbolischen Ebene, und das Ergebnis wird gemäß der entsprechenden Datendefinition aus dem Quellprogramm aufbereitet.

Bei jeder 4 Bytes langen Adresse, die zur indirekten Adressierung verwendet wird, berücksichtigt AID den aktuellen Adressierungsmodus des Testobjekts. Er ist mit %DISPLAY %AMODE abfragbar. Mit %AINT kann für die Pointer-Operation eine andere Adress-Interpretation vereinbart werden.

## Pointer-Operator

indirekte-adressierung mit Pointer-Operator - - - - -

$$\left\{ \begin{array}{l} \text{adresskonstante} \\ \text{speicherreferenz } [\%A[\text{Ln}] \mid \%S \mid \%SX] \end{array} \right\} \rightarrow \left[ \begin{array}{l} \text{strukturkomponente} \\ \text{BASED-variable} \end{array} \right]$$

-> Pointer-Operator

### adresskonstante

Anweisungsnamen, Source-Referenzen und das Ergebnis einer Adressselektion sind Adresskonstanten. Namen von Marken müssen vor "->" in L'...' gesetzt werden.

### speicherreferenz

kann jede Speicherstelle sein, die eine Adresse enthält. Daten vom Typ Adresse können Sie ohne Typmodifikation verwenden.

### [%A[Ln] | %S | %SX]

Typmodifikation, mit der eine Speicherstelle als Adresse interpretiert werden kann. Mit %S und %SX können Sie die Adressbildung von Maschinenbefehlen nachvollziehen. Aus Basisregister und Distanz (%S) oder Indexregister, Basisregister und Distanz (%SX) errechnet AID die Adresse wie die Hardware (siehe [Abschnitt „Speichertypen zur Interpretation von Maschinenbefehlen“ auf Seite 128](#)).

$$\left\{ \begin{array}{l} \text{strukturkomponente} \\ \text{BASED-variable} \end{array} \right\}$$

In diesen beiden Fällen wird der Pointer-Operator verwendet, um eine indirekte Adressierung nachzuvollziehen, die zum Sprachumfang der Programmiersprache gehört, nämlich um in C++/C eine Strukturkomponente über einen Zeiger zu referenzieren oder um in PL/I eine BASED-Variable über den zugehörigen Pointer anzusprechen. Für das Ergebnis der Indirektion gelten die im Quellprogramm definierten Attribute der Strukturkomponenten bzw. BASED-Variablen.

## Inhaltsoperator

In C++/C können Sie zur Dereferenzierung statt des Pointer-Operators auch den Inhaltsoperator verwenden.

Die mit dem Inhaltsoperator referenzierte Adresse wird entsprechend ihrem im Programm vereinbarten Datentyp interpretiert. Sie wechseln also nicht auf Maschinencode-Ebene wie dies bei unärer Dereferenzierung durch den Pointer-Operator geschieht. Im Gegensatz zu C++/C, wo Sie den Inhaltsoperator auch auf Vektoren anwenden können, ist der Inhaltsoperator in AID nur für Pointer zugelassen.

```
indirekte-adressierung mit Inhaltsoperator - - - - -
[[]* {...} pointer-variable[]]
```

\* Inhaltsoperator

pointer-variable

typbezogener Zeiger eines C++/C-Programms

Der Inhaltsoperator kann mehrfach wiederholt werden. Gegebenenfalls müssen Sie die Reihenfolge der Verarbeitung durch Klammerung festlegen. Der Inhaltsoperator wird nachrangig nach Pointer-Operator, Adressversatz und Indizierung ausgewertet wird.

Auf indirekte Adressierung kann folgen:

- Adressversatz "."
- indirekte Adressierung (->)
- Längenmodifikation
- Typmodifikation

## Beispiele

1. %DISPLAY V'10A', V'10A' ->

AID-Ausgabe

```
V'0000010A' = ABSOLUTE + #'0000010A'
0000010A (0000010A) 00000478      ....

V'00000478' = ABSOLUTE + #'00000478'
00000478 (00000478) E3C5E7E3      TEXT
```

AID gibt vier Bytes ab Adresse V'10A' im Format DUMP aus. Zur Ausgabe des zweiten Operanden verwendet AID diesen Speicherinhalt (X'00000478') als Adresse in einer Pointer-Operation und gibt vier Bytes ab Adresse V'478' im Format DUMP aus.

2. %FIND C'\*\*\*'  
%DISPLAY %1G->

Mit %FIND suchen Sie die Zeichenfolge '\*\*\*' im Speicher. Wenn AID die Zeichenfolge findet, wird im AID-Register %1G die Fortsetzungsadresse abgelegt, das ist die Adresse des ersten Bytes nach der gefundenen Zeichenfolge. Mit dem %DISPLAY-Kommando lassen Sie sich den auf den Suchbegriff folgenden Speicherinhalt ausgeben.

- 3.

```
%SET %7 INTO V'14C0'%SX->
Inhalt von Mehrzweckregister 4:      X'00000100'
Inhalt von Mehrzweckregister 6:      X'00004000'
Speicherinhalt ab Adresse V'14C0':  X'50746B00'  $\hat{=}$  ST R7,X'B00'(R4,R6)
```

AID simuliert eine Übertragung mit dem Befehl Speichern (ST, Befehlscode X'50') und überträgt den Inhalt von Mehrzweckregister 7 ab Adresse V'4C00'. Die Adresse errechnet sich AID aus dem Speicherinhalt X'50746B00' wie folgt:

```
X'507' wird ignoriert
X'4' nimm Inhalt von Register 4:      '00000100'
X'6' addiere Inhalt von Register 6:   '00004000'
X'B00' addiere Distanz:               'B00'
-----
ergibt die Adresse                    '4C00'
```

4. %SET X'C1C2C3C4' INTO V'14C2'%S->  
Register- und Speicherinhalte sind dieselben wie in Beispiel 3.  
Mit diesem %SET überträgt AID das Sedezimal-Literal X'C1C2C3C4' auf die Speicherstelle mit der Adresse V'4B00'. Die Adresse errechnet sich AID aus dem Speicherinhalt X'6B00' wie folgt:

```
X'6' nimm Inhalt von Register 6:      '00004000'
X'B00' addiere Distanz:               'B00'
-----
ergibt die Adresse                    '4B00'
```

### 7.2.4.3 Typmodifikation

Die Typmodifikation verwenden Sie, um einen Speicherinhalt anders interpretieren zu lassen als sein Speichertyp-Attribut vorgibt. Das kann in den folgenden Fällen notwendig sein:

- Anpassung der Typen beim %SET
- anderes Ausgabeformat bei %DISPLAY
- Konvertierung eines Literals (nur bei %DISPLAY zugelassen)
- Interpretation als Adresse vor einem Pointer-Operator
- Interpretation bzw. Aufbereitung in einer anderen Struktur (Redefinition einer Speicherstelle)
- Interpretation als Ganzzahl in einem Ausdruck.

Außer vor einem Pointer-Operator ist die Typmodifikation nur am Ende einer komplexen Speicherreferenz sinnvoll, um den Speicherinhalt ab der errechneten Adresse oder das Literals im gewünschten Speichertyp zu interpretieren.

typmodifikation

```

{ { speicherreferenz } { %typ[L-mod] } }
{ { literal %typ } { %T([bereichs-qua•]datename) } }

```

speicherreferenz

kann jede, wie auch immer angesprochene Speicherstelle sein: virtuelle Adresse, Datename, Schlüsselwort, C-Qualifikation

literal

Die AID-Literale sind ab [Seite 113](#) beschrieben.

%type[L-mod]

Schlüsselwort für Speichertypen mit wahlweiser Längenangabe:

%X, %C, %E, %P, %D, %F, %A und %UTF16 (siehe [Kapitel „Schlüsselwörter“ auf Seite 125](#)).

Die Länge können Sie mit allen Möglichkeiten der Längenmodifikation angeben. Ohne Längenangabe bleibt das Längenattribut des modifizierten Speicherobjekts erhalten.

Die Speichertypen %H, %Y, %S, %SX haben eine feste Länge. Sie können deshalb nur ohne Längenangabe verwendet werden.

Beim Speichertyp %UTF16 muss die Länge ein Vielfaches von 2 sein.

Die Längenmodifikation ist bei Literalen nicht zugelassen.

`%T([bereichs-qua.]datenname)`

Mit dem Typselektor verwenden Sie Speichertyp und Länge von anderen Datendefinitionen zur Interpretation einer Speicherstelle. Sie müssen deshalb auch die Regeln der Längenmodifikation beachten, z.B. Überschreiten der Bereichsgrenzen.

*datenname* kann qualifiziert sein, d.h. aus einem anderen Programmteil stammen. Eine Basisqualifikation können Sie jedoch nicht angeben.

Bei der Typmodifikation überprüft AID, ob der Speicherinhalt zu dem gewählten Speichertyp passt. Andernfalls gibt AID eine Fehlermeldung aus.

Jedem Speichertyp ist ein Ausgabetypp zugeordnet (siehe [Kapitel „Schlüsselwörter“ auf Seite 125](#)). Sie können deshalb mit der Typmodifikation auch den Ausgabetypp verändern.

Die Speichertypen `%D`, `%P`, `%F` und `%A` haben nur bestimmte zugelassene Längen (siehe [Kapitel „Schlüsselwörter“ auf Seite 125](#)). Wenn sie ohne Längenangabe eingesetzt werden, muss die Länge des modifizierten Speicherobjekts mit einer der zugelassenen Längen des Speichertyps übereinstimmen. Andernfalls weist AID eine Typmodifikation mit einem Längenfehler ab.

Die Speichertypen `%S`, `%H` und `%Y` haben die feste Länge 2 Bytes, `%SX` hat die feste Länge 4 Bytes. Mit ihnen wird zugleich eine Längenmodifikation ausgeführt. Die Längenmodifikation muss innerhalb der Bereichsgrenzen stattfinden können.

Die Typmodifikation mit dem Speichertyp `%UTF16` ist erlaubt, wenn die (implizite) Länge der Speicherstelle ein Vielfaches von 2 ist.

Die Typmodifikation `%UTF16` ist auf X-Literalen erlaubt, aber auf C- und U-Literalen verboten.

Da das `%SET`-Kommando bei der Übertragung den Typ und die Länge berücksichtigt und gegebenenfalls vor einer numerischen Übertragung den Speicherinhalt des Sendefeldes in den des Empfangsfeldes konvertiert, müssen die Datentypen von Sendefeld und Empfangsfeld verträglich sein. Dazu finden Sie eine Tabelle in der `%SET`-Beschreibung der sprachspezifischen Handbücher [2]-[6]. Sind die Datentypen nicht vereinbar, so können Sie einen zum Speicherinhalt passenden und mit dem Empfangsfeld verträglichen Speichertyp angeben.

Für Programmiersprachen, die die Definition von Strukturen vorsehen, gilt bezüglich des `%SET` die folgende Einschränkung: Strukturen können Sie mit einem `%SET`-Kommando nur modifizieren, wenn Sendefeld und Empfangsfeld dieselbe Struktur haben. Wurde eine der Adressen beim Schreiben des Programms nicht als Struktur beschrieben, so können Sie ihr mit der Typselektion die erforderliche Struktur zuweisen. Dann muss allerdings der aktuelle Speicherinhalt der Definition dieser Struktur entsprechen.



## Beispiele

1. `%DISPLAY V'10A'%F`  
Der Speicherinhalt ab Adresse V'10A' wird als Binärwert mit Vorzeichen interpretiert und als Ganzzahl mit Vorzeichen ausgegeben. Ohne die Typmodifikation hätte die virtuelle Adresse den Speichertyp Sedezimal (%X) und die Länge 4 Bytes und damit verbunden den Ausgabety DUMP.
2. `%INSERT V'4710'%SX->`  
Der Speicherinhalt der Adresse V'4710' wird zur Berechnung des Testpunktes entsprechend dem %SX-Format ausgewertet.
3. `%SET RECORD.10%PL5 INTO AMOUNT`  
In einem COBOL-Programm gibt es ein Datenfeld RECORD mit einer Länge von 45 Bytes, das eine Folge von gepackten Zahlen mit je 5 Bytes Länge enthält. AMOUNT ist ein numerisches, entpacktes Datenelement. Die ersten beiden Zahlen werden mit dem Adressversatz übersprungen. Durch die Typ- und Längenmodifikation wird die dritte gepackte Zahl aus RECORD entpackt und rechtsbündig nach AMOUNT übertragen.
4. `%D V'134'.(INDEX * 4)%T(LINE)`  
Zur virtuellen Adresse V'134' werden durch Adressversatz soviel Bytes dazugerechnet, wie sich aus dem Inhalt von INDEX multipliziert mit 4 ergeben. Der Speicherinhalt an der errechneten Adresse wird entsprechend dem Typ und der Länge der Datendefinition zu LINE aufbereitet und ausgegeben.
5. `%DISPLAY %1%F`  
Ohne Typmodifikation würde AID den Inhalt des Registers 1 als Sedezimalzahl ausgeben. Mit der Typmodifikation %F bereitet AID den Registerinhalt vor der Ausgabe als Ganzzahl mit Vorzeichen auf.
6. `%DISPLAY X'20AC'%UTF16`  
Durch die Typmodifikation erfolgt die Ausgabe im Dump-Format, d.h. neben dem Sedezimal-Code 20AC wird die Interpretation als UTF16-Code ausgegeben, in diesem Fall das Eurozeichen.

#### 7.2.4.4 Längenmodifikation

Mit der Längenmodifikation weichen Sie von der Länge einer Speicherreferenz ab. AID greift dann nicht auf die im Längenattribut hinterlegte Länge zu, sondern übernimmt die von Ihnen angegebene. Der Wert einer Längenmodifikation muss zwischen 1 und 65 535 liegen.

Ohne Angabe von *typ* beinhaltet die Längenmodifikation eine Typmodifikation in den Speichertyp %X.

Mit der Längenmodifikation dürfen die Bereichsgrenzen des modifizierten Speicherobjekts nicht überschritten werden, d.h. mit der neugebildeten Länge dürfen Sie die Endadresse nicht überschreiten.

Längenmodifikation - - - - -

$$\text{speicherreferenz } \%[\text{typ}] \left\{ \begin{array}{l} \%L_n \\ L(\text{speicherreferenz}) \\ L=(\text{ausdruck}) \end{array} \right\}$$

- - - - -

speicherreferenz

kann jede, wie auch immer angesprochene Speicherstelle sein: virtuelle Adresse, Datename, Schlüsselwort, C-Qualifikation oder komplexe Speicherreferenz

typ

Wenn Sie eine Typ- und Längenmodifikation durchführen wollen, geben Sie ein Schlüsselwort für Speichertypen an (%X, %C, %P, %D, %F, %A, %UTF16) und danach das L ohne weiteres Prozentzeichen.

*Beispiel:*

VAR1%XL5 oder VAR1%CL5

%Ln

Eine Längenmodifikation, die mit %L beginnt, beinhaltet zugleich eine Typmodifikation in den Standard-Speichertyp %X.

*n* ist eine positive Ganzzahl oder Sedezimalzahl mit  $0 \leq n \leq 65\,535$ , entsprechend dem zulässigen Wert einer Längenmodifikation.

%L(speicherreferenz)

Mit dem Längenselektor verwenden Sie das Längenattribut einer anderen Speicherreferenz für die Längenmodifikation. Der Längenselektor ist sinnvoll bei Datennamen, C- und COM-Qualifikationen.

**%L=(ausdruck)**

Mit der Längenfunktion lassen Sie sich die Länge von AID errechnen. (*ausdruck*) ist in [Abschnitt „Adress-, Typ- und Längenselektor“ auf Seite 102](#) beschrieben. Er wird gebildet aus Ganzzahlen, Inhalten von Speicherreferenzen, die vom Typ Integer (%F oder %A) und einer Länge  $\leq 8$  sein müssen, dem Ergebnis von Adressselektor, Längenselektor und Längenfunktion und den arithmetischen Operatoren (+ - \* /).

Die beteiligten Operanden und das resultierende Ergebnis müssen im Wertebereich eines %FL8 Feldes liegen.

Steht im *ausdruck* einer Längenfunktion nur eine Speicherreferenz, nimmt AID den Inhalt, nicht die Länge, als Wert für die Längenmodifikation.

Es wird der Wertebereich  $-2^{63} \leq n < +2^{64}$  unterstützt. Damit können die Datentypen %FL8 mit den Werten  $-2^{63} \leq n < +2^{63}$  und %AL8 mit den Werten  $0 \leq n < +2^{64}$  korrekt dargestellt werden. Wenn das Ergebnis nicht in den Wertebereich passt, kommt es zu einer Fehlermeldung AID0470.

Verwenden Sie einen Längenselektor auf einen Vektor, ohne einen Index dazu anzugeben, wird die Länge des gesamten Vektors selektiert. Nur mit Angabe des Index kann AID auf die Länge eines Elements des Vektors zugreifen.

## Beispiele

1. %DISPLAY V'10A'%L=(VAR1)

VAR1 ist vom Typ Integer und enthält den Wert 23. Ab der Adresse V'10A' werden 23 Bytes im Ausgabeformat DUMP ausgegeben.

2. %SET CVAR1%CL(CVAR) INTO CVAR

CVAR1 und CVAR seien zwei Character-Variablen in einem Fortran-Programm. und CVAR1 sei länger als CVAR. Die Längenmodifikation ermöglicht es Ihnen, CVAR1 linksbündig in der Länge von CVAR zu übertragen.

3. %SET %L(CVAR) INTO %2G

Die Länge der Variablen CVAR wird in AID-Register %2G übertragen.

4. %DISPLAY V'10A'%AL3->

Der Inhalt der drei Bytes ab Adresse V'10A' wird als Adresse interpretiert. AID gibt ab der damit angesprochenen Speicherstelle 4 Bytes im Dump-Format (%XL4) aus.

5. %D V'10A'%L=(INDEX\*12-%L(NAME))

Hier ergibt sich die Länge aus der Multiplikation des Inhalts von INDEX mit 12. Davon wird Länge von NAME abgezogen.

6. %D V'4700'%L=(%L(C=CS1)-%L(INDAT))

Die Länge wird berechnet aus der Länge der CSECT CS1 abzüglich der Länge von INDAT.

7.2.4.5 **Arithmetischer Ausdruck**

Beim Adressversatz, in einer Längenfunktion und im Index können Sie einen arithmetischen Ausdruck schreiben, um von AID den benötigten Wert errechnen zu lassen. Einen Ausdruck können Sie also überall dort einsetzen, wo ein ganzzahliger Wert erforderlich ist. Die Rechenoperatoren werden von AID entsprechend den mathematischen Regeln bei der Auflösung eines arithmetischen Ausdrucks abgearbeitet. Durch Setzen von runden Klammern können Sie eine andere Reihenfolge der Abarbeitung festlegen. Vor und nach einem Minuszeichen "-" empfiehlt es sich, ein Leerzeichen zu schreiben, so dass auch bei der Einstellung %AID SYMCHARS[=STD] keine Fehlinterpretationen vorkommen können.

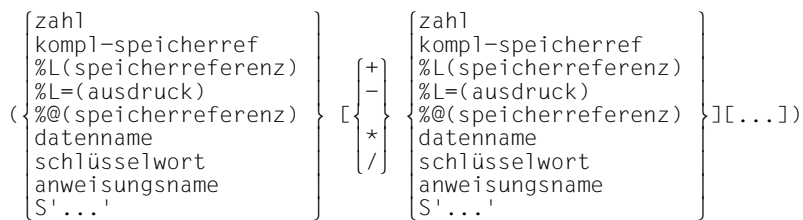
Für jeden Schritt der Abarbeitung von *ausdruck* gilt:  $-2^{63} \leq \text{Zwischenergebnis} \leq 2^{63}-1$

Für den Adressversatz gilt:  $-2^{31} \leq \text{Endergebnis} \leq 2^{31}-1$

Für die Längenfunktion gilt:  $0 \leq \text{Endergebnis} \leq 65\ 535$

Für den Index gelten die im Quellprogramm festgelegten Grenzwerte. Außerdem können nur die Operanden *zahl* und *datenname* verwendet werden (siehe [Abschnitt „Datennamen“ auf Seite 82](#)).

ausdruck



*zahl*

ist eine Ganzzahl oder eine Sedezimalzahl mit  $-2^{63} \leq \text{zahl} \leq 2^{63}-1$ .

*kompl-speicherref*

kann jede, wie auch immer angesprochene Speicherstelle sein. Ihr Inhalt muss ganzzahlig sein, also vom Typ %F oder %A mit einer Länge  $\leq 8$ .

Auf diese Art können Sie den Inhalt einer Speicherreferenz für einen Adressversatz, zur Längenmodifikation oder als Index/Subskript einsetzen.

%L(speicherreferenz)

Mit dem Längenselektor greifen Sie auf das Längenattribut einer Speicherreferenz zu. Das Ergebnis ist ein ganzzahliger Wert. Der Längenselektor ist sinnvoll bei Datennamen, C- und COM-Qualifikationen. Die Länge von anderen Speicherreferenzen, z.B. Schlüsselwörtern, ist ja bekannt.

%L=(ausdruck)

Mit der Längenfunktion lassen Sie sich von AID einen ganzzahligen Wert errechnen. Ausdruck entspricht den hier beschriebenen Regeln.

**%@(Speicherreferenz)**

Mit dem Adressselektor greifen Sie auf das Adressattribut einer Speicherreferenz zu. Das Ergebnis ist eine Adresskonstante (%AL4).

**datenname**

muss im Quellprogramm vom Typ Integer oder Adresse mit einer Länge  $\leq 8$  definiert sein. Der Inhalt von *datenname* wird zur Berechnung des arithmetischen Ausdrucks herangezogen.

**schlüsselwort**

Der Inhalt von *schlüsselwort* wird zur Berechnung des arithmetischen Ausdrucks verwendet. Sie können die folgenden Schlüsselwörter (siehe [Kapitel „Schlüsselwörter“ auf Seite 125](#)) angeben:

%n	Mehrzweckregister, $0 \leq n \leq 15$
%nG	AID-Mehrzweckregister, $0 \leq n \leq 15$
%PC	Befehlszähler (Program Counter)
%•[subkdoname]	Durchlaufzähler. Mit der Kurzform %• bezeichnen Sie den Durchlaufzähler des gerade aktiven Subkommandos.

**anweisungsname**

Da Anweisungsnamen Adresskonstanten sind, können sie in einem Ausdruck eingesetzt werden.

**S'...'**

Source-Referenzen sind ebenfalls Adresskonstanten. Auch sie können in einem Ausdruck eingesetzt werden.

**Beispiele**

1. `%D %L=(%1+5)`  
Die Länge, die sich aus dem Inhalt von Register %1 erhöht um 5 ergibt, wird ausgegeben.
2. `%D V'0'.(V'100'%AL2 + %L(C=CSECT))`  
Ab Adresse V'0' wird ein Adressversatz in der Länge des in Klammern angegebenen Ausdrucks durchgeführt. Zunächst wird der Inhalt der beiden Bytes mit den Adressen V'100' und V'101' als positive Ganzzahl in der Länge 2 interpretiert. Anschließend wird die Länge von CSECT dazuaddiert. Ab der so errechneten Speicherstelle gibt AID 4 Bytes im Dump-Format aus.
3. `%S %L=((V'0'%AL4 + V'4'%AL1) * NUM1) INTO %2G`  
Der Wert, den die Längenfunktion errechnet, wird in AID-Register %2G übertragen. V'0' enthält X'00000005', V'4' enthält X'FFF5003A', und NUM1 enthält den Wert 3. Das ergibt nach den Typ- und Längenmodifikationen:  $(5 + 255) * 3 = 780$ . Der Wert 780 wird in %2G übertragen.

### 7.2.4.6 Adress-, Typ- und Längenselektor

Mit den Selektoren greifen Sie auf die Attribute einer Speicherreferenz zu.

selektoren - - - - -

```
%@(speicherreferenz)
%T([bereichs-qua.]datenname)
%L(speicherreferenz)
```

- - - - -

#### %@(speicherreferenz)

Mit dem Adressselektor greifen Sie auf das Adressattribut einer Speicherreferenz zu. Das Ergebnis ist eine Adresskonstante (%AL4). Sie kann vor einem Pointer-Operator zum Zugriff auf eine Speicherstelle oder als Binärzahl ohne Vorzeichen in einem Ausdruck eingesetzt werden. Mit %DISPLAY können Sie das Ergebnis der Adressselektion ausgeben lassen.

Der Adressselektor ist sinnvoll bei Datennamen, C- und COM-Qualifikationen. Die Adressen von Schlüsselwörtern, die außerhalb des Benutzerbereichs liegen, können Sie sich zwar mit %DISPLAY ausgeben lassen, jedoch nicht mit einem nachfolgenden Pointer-Operator als Speicherreferenz verwenden.

#### %T([bereichs-qua.]datenname)

Mit dem Typselektor greifen Sie auf das Typ- und Längenattribut einer Speicherreferenz zu. Der selektierte Datentyp kann nur zur Typmodifikation eingesetzt werden. *datenname* kann qualifiziert sein.

#### %L(speicherreferenz)

Mit dem Längenselektor greifen Sie auf das Längenattribut einer Speicherreferenz zu. Das Ergebnis ist eine positive Ganzzahl. Sie kann zur Längenmodifikation oder in einem Ausdruck eingesetzt werden. Mit %DISPLAY können Sie das Ergebnis der Längenselektion ausgeben lassen.

Der Längenselektor ist sinnvoll bei Datennamen, C- und COM-Qualifikationen. Die Länge von anderen Speicherreferenzen, z.B. Schlüsselwörtern und virtuellen Adressen, ist ja bekannt.

## Beispiele

- ```
%D @(VAR)
%D @(VAR)->.8
%S V'2E'.( @(VAR))%CL2 INTO X
%D V'A1A'%XL=(2+@(VAR))
```

**Einsatzmöglichkeiten des Adressselektors:**

Im ersten %DISPLAY wird eine Adresse ausgegeben.

Im zweiten %DISPLAY wechseln Sie mit Adressselektion und Pointer-Operator auf Maschinencode-Ebene, um beim Adressversatz nicht durch die Bereichsgrenzen von VAR behindert zu werden.

Im %SET benutzen Sie die Adresse von VAR als Wert für einen Adressversatz.

Im letzten %DISPLAY benutzen Sie den Wert der Adresse von VAR, um die Länge zu errechnen.

- ```
%D V'100'%T(VAR)
%S V'100'%T(INT) INTO NUM1
```

**Einsatzmöglichkeiten des Typselektors:**

Im %DISPLAY wird der Inhalt einer virtuellen Adresse in Typ und Länge von VAR ausgegeben (Redefinition).

Im %SET wird der Inhalt einer virtuellen Adresse im Typ und der Länge der Integervariablen INT interpretiert, damit er werterhaltend in die numerische Variable NUM1 übertragen werden kann.

- ```
%D %L(VAR)
%S %L(VAR) INTO NUM1
%D V'A1A'.(2+%L(VAR))
%D V'A1A'%L=(%L(VAR)*5)
```

**Einsatzmöglichkeiten des Längenselektors:**

Im ersten %DISPLAY wird die Länge von VAR ausgegeben.

Im %SET wird der Wert der Länge von VAR in die numerische Variable NUM1 übertragen.

Im zweiten %DISPLAY benutzen Sie die Länge von VAR als Wert in einem Adressversatz-Ausdruck.

Im letzten %DISPLAY benutzen Sie den Wert der Länge von VAR in einer Längenmodifikation.

### 7.2.4.7 Besonderheiten beim Zusammenwirken der verschiedenen Bestandteile

Beginnt eine komplexe Speicherreferenz mit einer Adresskonstanten (z.B. mit einer Source-Referenz oder einer Marke), muss anschließend der Pointer-Operator geschrieben werden. Namen von Marken müssen dabei stets in L'...' gesetzt werden. Ohne den Pointer-Operator können Adresskonstanten innerhalb der *kompl-speicherref* überall da stehen, wo auch Sedezimalzahlen geschrieben werden können. Nach Adressversatz oder Pointer-Operation (Ausnahmen bei C++/C und PL/I, siehe [Abschnitt „Adressversatz“](#) auf Seite 89 und [Abschnitt „Indirekte Adressierung“-> / "\\*/" auf Seite 91](#)) gehen impliziter Speichertyp und implizite Länge der Ausgangsadresse verloren. Falls Sie nicht explizit einen anderen Speichertyp und eine andere Länge vereinbaren, gilt an der errechneten Stelle der Speichertyp %X in der Länge 4, außer Sie setzen die komplexe Speicherreferenz als Empfänger im Kommando %MOVE ein. In diesem Fall geht der Bereich, der mit %MOVE überschrieben werden darf, von der Anfangsadresse von *kompl-speicherref* bis zum Ende des von Ihrem Programm belegten Speichers.

Für keinen Operanden in einer komplexen Speicherreferenz darf der zugeordnete Speicherbereich durch einen Adressversatz oder eine Längenmodifikation überschritten werden, sonst schreibt AID eine Fehlermeldung. Wenn Sie jedoch die Anfangsadresse eines Speicherobjekts nutzen wollen, ohne auf die Bereichsgrenzen achten zu müssen, verwenden Sie die Adresselektion in Verbindung mit dem Pointer-Operator (%@(...)->). Sie verlassen damit die symbolische Ebene, was gleichzeitig beinhaltet, dass Sie auf das Typ- und Längenattribut des angesprochenen Objekts dann nur noch über die entsprechenden Selektoren zugreifen können.

Manche Compiler, wie C++/C und PL/I, erzeugen bei der Übersetzung zu jedem Programm oder Unterprogramm einen Prolog. Mit *funktion* (C++/C) bzw. *entry* (PL/I) ohne anschließenden Pointer-Operator bezeichnen Sie die erste ausführbare Anweisung der entsprechenden Funktion oder Prozedur. Wenn Sie jedoch an *funktion* oder *entry* den Pointer-Operator anschließen, um vom Funktions- oder Prozeduranfang ausgehend weiterzupositionieren, müssen Sie beachten, dass die Adressrechnung dann mit der Anfangsadresse des Prologs beginnt.

### Zeichencodierung einer Zeichenfolge

Mit den Konvertierungsfunktionen %C() und %UTF16() kann die Interpretation der Zeichenfolgen-Codierung verändert werden. Die Speicherreferenz muss vom Typ „Zeichenfolge“ sein, also z.B. vom Typ %C oder %UTF16.

Die Konvertierungsfunktionen haben nur dann eine Wirkung, wenn %C() auf den Typ %UTF16 oder %UTF16() auf den Typ %C angewandt wird. Hierbei bleiben die Speicherstellen unverändert. AID arbeitet implizit mit der konvertierten Speicherstelle weiter.

Der CCSN für den Typ %C wird aus der %AID EBCDIC Einstellung verwendet. Die aktuell gültigen Einstellungen können mit dem Kommando %SHOW %AID angezeigt werden.



**Beispiel:**

An der Speicherstelle V'00' steht das Byte X'BB' .

1. %AID EBCDIC=EDF03IRV  
Das Byte X'BB' beschreibt demzufolge das Zeichen 'I' im CCSN EDF03IRV.  
%UTF16(V'00' %CL1) ergibt den Hexadezimalwert X'005B'.
2. %AID EBCDIC=EDF03DRV  
Das Byte X'BB' beschreibt das Zeichen C'Ä' im CCSN EDF03DRV  
%UTF16(V'00' %CL1) ergibt den Hexwert X'00C4'.



---

## 8 Operand Medium-und-Menge

Der Operand *medium-u-menge* legt fest, mit welchem Ausgabemedium AID arbeiten soll und ob es außer dem Inhalt des bezeichneten Speicherbereichs (Daten) Zusatzinformationen ausgeben soll.

Dieser Operand kann, durch Komma getrennt, mehrfach angegeben werden, so dass z.B. mit T=MIN, P=MAX minimaler Informationsumfang am Terminal und maximaler Informationsumfang auf SYSLST ausgegeben wird.

Den Operanden *medium-u-menge* können Sie in den folgenden Kommandos angeben:

%DISPLAY

%HELP

%SDUMP

%OUT

Der *medium-u-menge*-Operand des %OUT-Kommandos wirkt außerdem auf:

%DISASSEMBLE

%TRACE

Mit dem Kommando %OUT kann *medium-u-menge* für die Kommandos %DISASSEMBLE, %DISPLAY, %HELP, %SDUMP und %TRACE festgelegt werden. Die Vereinbarung gilt für die gesamte Testsitzung bis zu einer neuen Vereinbarung oder der Beendigung mit /EXIT-JOB.

In %DISPLAY, %SDUMP und %HELP kann *medium-u-menge* direkt angegeben werden; die Vereinbarung gilt nur für das jeweilige Kommando, danach tritt wieder *medium-u-menge* des Kommandos %OUT oder der Standardwert T=MAX in Kraft. Wird in %DISPLAY, %SDUMP oder %HELP kein *medium-u-menge* angegeben, so gilt der *medium-u-menge* des %OUT-Kommandos. Wurde auch in %OUT *medium-u-menge* nicht angegeben, so gilt der Standardwert T=MAX.

Für alle Ausgabemedien berücksichtigt AID den zugeordneten Coded-Character-Set Namen (kurz CCSN) immer dann, wenn UTF16-/ UTFE-Zeichen auf das Ausgabemedium ausgegeben werden sollen.

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

$$\left. \begin{array}{l} \text{I} \\ \text{H} \\ \text{Fn} \\ \text{P} \end{array} \right\} = \left. \begin{array}{l} \text{MIN} \\ \text{MAX} \\ \text{XMAX} \\ \text{XFLAT} \end{array} \right\}$$

- - - - -  
**I** Terminal-Ausgabe über SYSOUT.

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

**Fn** Datei-Ausgabe. Fn bezeichnet den Linknamen für die Ausgabedatei.  
n ist eine Zahl mit einem Wert  $0 \leq n \leq 7$ .

Es gibt drei Wege, die zugehörige Datei anzulegen bzw. die Ausgabedatei zuzuweisen:

1. %OUTFILE-Kommando mit dem Link- und Dateinamen.
2. ADD-FILE-LINK-Kommando für Fn.
3. Für einen Linknamen, dem kein Dateiname zugewiesen ist, setzt AID einen FILE-Makro mit dem Dateinamen AID.OUTFILE.Fn ab, entsprechend dem Linknamen Fn. Die Datei wird mit FCBTYP=SAM, OPEN=EXTEND und RECFORM=V erstellt.

Beachten Sie bei der Verwendung des Linknamens F6, dass F6 der Standard-Linkname für REP-Dateien ist.

**P** Ausgabe auf SYSLST.

### MAX

Die maximalen Zusatzinformationen werden mit den Daten ausgegeben. Das sind zum einen Informationen über den AID-Arbeitsbereich und über die Unterbrechungsstelle, zum anderen Informationen über die auszugehenden Daten.

Bei %HELP hat die Angabe {MIN | MAX} keine Auswirkungen, eine der beiden Angaben ist aber syntaktisch erforderlich.

Wenn für ein Ausgabemedium zum ersten Mal eine Ausgabe mit dem Operandenwert MAX durchgeführt wird bzw. wenn sich der Zeileninhalt gegenüber einer vorhergegangenen Ausgabe geändert hat, werden der Ausgabe bis zu drei Zeilen vorgestellt:

- Taskzeile, sie informiert über den aktuellen AID-Arbeitsbereich und enthält den Task Identifier (TID) sowie die Prozessfolge-Nummer (TSN) oder den Linknamen der Dump-Datei.
- Kopfzeile, sie enthält Informationen über die Unterbrechungsstelle, das ist die Adresse, an der das Programm zum Zeitpunkt der Ausgabe steht.
- Zielzeile, sie enthält Informationen über die auszugebende Adresse, das ist die CSECT/der COMMON, in der sich die Adresse befindet, und die Distanz zum CSECT-/COMMON-Anfang.

Für jeden Datenbereich werden ausgegeben:

- Datenname und bei Vektoren auch die Indexgrenzliste
- Inhalt und bei Vektoren auch der zugehörige Index; bei gleichen Zeilen wird der Text "repeated lines: n" ausgegeben.
- virtuelle Adresse des ersten Bytes einer jeden Datenzeile bei Ausgaben auf Maschinencode-Ebene
- Distanz des ersten Bytes einer jeden Datenzeile zum Beginn einer CSECT, falls die Adresse einer CSECT zugeordnet werden kann; ansonsten die Adresse des ersten Bytes einer jeden Datenzeile relativ zum Anfang des Datenbereiches.

MIN Daten werden ohne Zusatzinformationen ausgegeben.

XMAX Die Datenausgabe mit dem Kommando %SDUMP ist annähernd identisch zur Datenausgabe beim Standardwert MAX mit folgendem Unterschied: Jedem Datenelement geht ein Typ-Tag voraus, das Typ, Größe und Ausgabe-Format dieses Datelements definiert.

In den Kommandos %DISPLAY, %DISASSEMBLE, %HELP and %TRACE wird der Operandenwert XMAX nicht berücksichtigt. Die Standard-MAX-Ausgabe wird generiert.

XFLAT Die Datenausgabe mit dem Kommando %SDUMP ist annähernd identisch zur Datenausgabe beim Wert XMAX mit folgendem Unterschied: Für strukturierte Datentypen wird nur die jeweils oberste Strukturebene ausgegeben. Bei langen Daten (z.B. langen Strings oder Arrays) werden nur die ersten Elemente ausgegeben.

In den Kommandos %DISPLAY, %DISASSEMBLE, %HELP and %TRACE wird der Operandenwert XFLAT nicht berücksichtigt. Die Standard-MAX-Ausgabe wird generiert.

## Beispiele

```
%OUT %DA T=MIN
%CI %IO <%DA FROM %PC->;%STOP>
%R
00000BB6 L R1,A8(R0,R11)
00000BBA L R15,98(R0,R11)
00000BBE BALR R14,R15
00000BC0 DC X'0001' INVALID OP CODE
00000BC2 CLI 396(R12),X'F0'
00000BC6 L R13,B4(R0,R2)
00000BCA BC B'1100',E0(R0,R13)
00000BCE L R15,A4(R0,R11)
00000BD2 BAL R14,4(R0,R15)
00000BD6 DC X'0000' INVALID OP CODE

STOPPED AT SRC REF: 540PE, SOURCE: TEST, PROC: TEST
```

Ohne Zusatzinformationen sollen die Daten, die das %DISASSEMBLE-Kommando ermittelt, ausgegeben werden. Die virtuelle Adresse des jeweiligen Befehls wird als 8-stellige Sedezimal-Zahl vorangestellt.

```
%OUT %DA T=MAX
%CI %IO <%DA FROM %PC->;%STOP>
%R
TEST+C22 L R1,A8(R0,R11) 58 10 B0A8
TEST+C26 L R15,9C(R0,R11) 58 F0 B09C
TEST+C2A BALR R14,R15 05 EF
TEST+C2C CLI 396(R12),X'F2' 95 F2 C396
TEST+C30 L R13,B4(R0,R2) 58 D0 20B4
TEST+C34 BC B'1100',14A(R0,R13) 47 C0 D14A
TEST+C38 L R15,A4(R0,R11) 58 F0 B0A4
TEST+C3C BAL R14,4(R0,R15) 45 E0 F004
TEST+C40 DC X'0000' INVALID OP CODE 00 00
TEST+C42 CLI 396(R12),X'F0' 95 F0 C396

STOPPED AT SRC REF: 58REA, SOURCE: TEST, PROC: TEST
```

Mit Zusatzinformationen sollen die Daten, die das %DISASSEMBLE-Kommando ermittelt, ausgegeben werden. Die Adresse des Befehls wird als relative Adresse angegeben, d.h. als Programmname plus Abstand zum Programmfang. Hinter den disassemblierten Befehlen steht der Speicherinhalt in sedezimaler Form.

```
%OUT %D T=MIN
%D ABC-TAB

01 ABC-TAB
02 ZEICHEN( 1: 26)
   |A| |B| |C| |D| |E| |F| |G| |H| |I| |J| |K| |L| |M|
   |N| |O| |P| |Q| |R| |S| |T| |U| |V| |W| |X| |Y| |Z|
```

Ohne Zusatzinformationen soll die Tabelle ABC-TAB aus einem COBOL-Programm mit %DISPLAY ausgegeben werden. Die Stufennummern und der Inhalt der Tabellenfelder werden ausgegeben.

```
%D ABC-TAB T=MAX  
  
*** TID: 000000D1 *** TSN: 8438 *****  
SRC_REF:      58ADD  SOURCE: MOBS      PROC: MOBS *****  
01      ABC-TAB  
02      ZEICHEN( 1: 26)  
        ( 1) |A| ( 2) |B| ( 3) |C| ( 4) |D| ( 5) |E| ( 6) |F|  
        ( 7) |G| ( 8) |H| ( 9) |I| (10) |J| (11) |K| (12) |L|  
        (13) |M| (14) |N| (15) |O| (16) |P| (17) |Q| (18) |R|  
        (19) |S| (20) |T| (21) |U| (22) |V| (23) |W| (24) |X|  
        (25) |Y| (26) |Z|
```

Mit Zusatzinformationen soll die Tabelle ABC-TAB aus einem COBOL-Programm mit %DISPLAY ausgegeben werden. Eine Task- und eine Kopfzeile wird der eigentlichen Ausgabe vorangestellt. Die Stufennummern, der Inhalt der Tabellenfelder und der zugehörige Index werden ausgegeben.





---

## 9 AID-Literale

In den AID-Kommandos %DISPLAY, %FIND, %MOVE und %SET ist als Operand die Angabe eines AID-Literals möglich.

### 9.1 Zeichen-Literale

#### 9.1.1 Character-Literal

##### 9.1.1.1 Eingabeformate

{C'x...x' | 'x...x'C | 'x...x'| U'x...x'}

maximale Länge: 80 Zeichen.

Zeichenvorrat für x: jedes Zeichen, das Sie am Terminal eingeben können.

Wenn der Coded-Character-Set für das Eingabemedium nicht UTFE ist, kann mit Hilfe des U-Literals ein UTFE-Character-String spezifiziert werden.

Kleinbuchstaben können nur eingegeben werden, wenn %AID LOW[=ON] eingestellt ist. Voreinstellung ist, dass Kleinbuchstaben in Großbuchstaben umgesetzt werden (siehe %AID). Dann können Sie Kleinbuchstaben nur als Sedezimal-Literale eingeben. Apostrophen, die im Literal enthalten sein sollen, müssen verdoppelt werden (").

**%DISPLAY**

Das Literal wird ausgegeben. Es kann u.U. über eine Typmodifikation konvertiert werden.

**%FIND**

x kann auch das Wildcard-Symbol '%' sein. Es steht jeweils für ein beliebiges Zeichen und wird von %FIND immer als Treffer gemeldet.

Bei Verwendung von %C() und %UTF16() auf das Suchliteral wird '%' nicht mehr als Wildcard-Symbol unterstützt.

**%MOVE**

Das Literal wird linksbündig in der Länge des Literals in das Empfangsfeld übertragen. Ist das Literal länger als das Empfangsfeld, wird die Übertragung mit einer Meldung abgewiesen.

### %SET

Besteht das Literal nur aus Ziffern, ist seine Länge  $\leq 18$  und soll es in ein numerisches Feld übertragen werden, so wird es wie ein numerisches Literal konvertiert und werterhaltend übertragen. Ist der Inhalt nicht rein numerisch oder seine Länge  $> 18$ , so kann das Literal alphanumerisch in ein Character-Feld (%C) oder binär in ein Feld mit Typmodifikation %X übertragen werden. Dort wird es linksbündig abgelegt. Ist das Empfangsfeld länger als das Literal, wird rechts aufgefüllt, bei alphanumerischer Übertragung mit Leerzeichen (C' ' ≐ X'40'), bei binärer Übertragung mit X'00'. Ist das Literal länger als das Empfangsfeld, wird rechts abgeschnitten und eine Warnung ausgegeben.

#### 9.1.1.2 Zeichencodierung

Mit der Einführung von Unicode unterstützt AID für Eingabe- und Ausgabemedien den diesen Medien zugeordneten **Code-Character-Set-Name** (CCSN).

Einer Datei kann mit dem Kommando MODIFY-FILE-ATTRIBUTES über den Operanden CODED-CHARACTER-SET ein CCSN zugewiesen werden.

Beim Eingabemedium TERMINAL benutzt AID den CCSN, der über das BS2000 Kommando MODIFY-TERMINAL-OPTIONS eingestellt wurde. Dabei sind AID alle Einstellungen unbekannt, die direkt an der Terminal-Emulation vorgenommen, aber nicht über das Kommando MODIFY-TERMINAL-OPTIONS bekannt gemacht wurden.

Wurden für Eingabe- oder Ausgabemedien keine CCSNs vereinbart, verwendet AID als Standard-CCSN den CCSN des Benutzerkatalog-Eintrages der Benutzerkennung. Diese Einstellung kann über das Kommando %AID EBCDIC= ... geändert werden.

AID unterstützt für Eingabe- und Ausgabemedien nur diejenigen CCSNs, die auch von XHCS unterstützt werden und außer UTFE einen 1-Byte-EBCDIC-Code repräsentieren.

Mit dem AID-Kommando %SHOW %CCSN können die aktuell von XHCS-SYS unterstützten CCSNs ausgegeben werden.

#### 9.1.1.3 Konvertierungsfunktion %C() und %UTF16()

Mit diesen Funktionen können sie die Art der Zeichencodierung eines Character-Literals verändern.

%UTF16() konvertiert das Literal in eine UTF16-Zeichenfolge.

%C() konvertiert ein UTF16-Literal in eine 1-Byte-EBCDIC-Codierung, die durch das Kommando %AID EBCDIC festgelegt wurde.

Liegt das Literal in einer 1-Byte-EBCDIC-Codierung vor, so hat %C() keine Wirkung.

Bei der Konversion wird ein Zeichen durch das Ersatzzeichen ' ' ersetzt, wenn es im UTF16 oder 1-Byte-EBCDIC-Zeichensatz nicht vorhanden ist. In diesem Fall gibt AID eine Meldung aus.

### 9.1.1.4 Suchen nach Character-Literalen mit %FIND

Damit das %FIND-Kommando Character-Literale finden kann, muss der CCS von *find-bereich* mit dem CCS des Eingabemediums (SYSCMD) übereinstimmen. Legen Sie daher den CCS von *find-bereich* fest, bevor Sie in *find-bereich* nach einem Character-Literal suchen:

```
%AID CCS= CCS-name
```

Üblicherweise ist EDF03IRV als CCS voreingestellt, es kann aber auch ein beliebiger anderer, in den Attributen der Benutzerkennung eingestellter, EBCDIC-Zeichensatz, wie z.B. EDF04F, sein.

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

#### Beispiel:

Der CCS von SYSOUT wird auf Unicode gesetzt:

```
/MODIFY-TERM-OPTION CODED-CHARACTER-SET=UTFE
```

```
FTEST: PROC OPTIONS(MAIN);
        /* UTFE */
DCL UTFED BIT(200) INIT('D49EB7D5C3C8C5D56DC4C5'X); /* DE
DCL UTFEB BIT(200)
        INIT('4541B745B0AF45AAB845B0B245AAA045AAB86DC2E8'X); /* 21
DCL CED CHAR(11) DEFINED UTFED;
DCL CEB CHAR(21) DEFINED UTFEB;
DCL CE CHAR(9) INIT('Munich_EN');
DCL SE CHAR(47);
        /* UTF-8 */
DCL UTF8D BIT(200) INIT('4DC3BC6E6368656E2D4445'X); /* DE 11
DCL UTF8B BIT(200) INIT('D09CD18ED0BDD185D0B5D0BDD2D4259'X); /* BY
DCL ISO BIT(200) INIT('4D756E6963682D454E'X); /* EN 9
DCL B8 BIT(16) INIT('2020'X);
DCL C8D CHAR(11) DEFINED UTF8D;
DCL C8B CHAR(15) DEFINED UTF8B;
DCL C8 CHAR(9) DEFINED ISO;
DCL BL CHAR(2) DEFINED B8;
DCL S8 CHAR(41);
        /* 8-bit ISO8859x */
DCL IS08D BIT(200) INIT('4DFC6E6368656E2D4445'X); /* DE 10
DCL IS08B BIT(200) INIT('BCEEDDE5D5DD2D4259'X); /* BY 9
DCL C7D CHAR(10) DEFINED IS08D;
DCL C7B CHAR(9) DEFINED IS08B;
DCL S7 CHAR(34);
        /* UTF-16 */
```

```

DCL U16D BIT(200) INIT('004D00FC006E006300680065006E003D00440045'X);
DCL U16B BIT(200) INIT('041C042E041D04250415041D003D00420059'X);
DCL U16 BIT(200) INIT('004D0075006E006900630068003D0045004E'X);
DCL B16 BIT(32) INIT('00200020'X);
DCL C16D CHAR(20) DEFINED U16D;
DCL C16B CHAR(18) DEFINED U16B;
DCL C16 CHAR(18) DEFINED U16;
DCL BB CHAR(4) DEFINED B16;
DCL S16 CHAR(68);

SE= CED ||' '|| CEB ||' '|| CE ||' ' ;
S8= C8D || BL || C8B || BL || C8 || BL ;
S7= C7D || BL || C7B || BL || C8 || BL ;
S16= C16D || BB || C16B || BB || C16 || BB;
STOP: ;
END FTEST;

```

Nach dem Laden des Programms:

```

%AID LOW=ON
%INSERT S=FTEST.STOP
%RESUME

```

Die Ausgabe der UTFE-Daten mit %DISPLAY:

```

/%D SE %X 'UTFE'
V'01028290' = SE          + #'00000000'
01028290 (00000000) D49EB7D5 C3C8C5D5 6DC4C540 404541B7      MÜNCHEN_DE ?
010282A0 (00000010) 45B0AF45 AAB845B0 B245AAA0 45AAB86D      Мюнхен_
010282B0 (00000020) C2E84040 D4A49589 83886DC5 D54040      BY Munich_EN

```

Die Suche mit %FIND nach 'Мюнхен' im UTFE-Bereich mit %FIND liefert kein Ergebnis:

```

/%FIND 'Мюнхен' IN SE
% AID0351 Kein Treffer im Suchbereich

```

Mit %SHOW wird der CCS von *find-bereich* (EDF03IRV) ermittelt:

```

/%SH %AID
%AID CHECK      = NO
%AID REP        = NO
%AID SYMCHARS   = STD
%AID OV         = NO
%AID LOW        = ON
%AID DELIM      = '| '|
%AID LANG       = D
%AID FORK       = NOT_USED

```

```

%AID EXEC      = OFF
%AID C         = NO
%AID EBCDIC    = EDF03IRV
%AID CCS       = EDF03IRV
%AID LEV       = OFF

```

Der CCS von *find-bereich* wird auf UTFE gesetzt:

```
/%AID CCS=UTFE
```

Die Suche im UTFE-Bereich mit %FIND liefert jetzt Ergebnisse:

```

/%FIND 'Мюнхен' IN SE
ABSOLUT +0102829D=0102829D : 4541B745 B0AF45AA B845B0B2 Мюнхен_

/%FIND 'E' IN SE
ABSOLUT +01028296=01028296 : C5D56DC4 C5404045 41B7 EN_DE M

/%F
ABSOLUT +0102829A=0102829A : C5404045 41B745B0 AF45AAB8 E Мюн

```

Der CCS von *find-bereich* wird auf UTF8 gesetzt:

```
/%AID CCS=UTF8
```

Der CCS von *find-bereich* wurde auf UTF8 gesetzt, aber %FIND ohne Parameter verwendet weiterhin CCS=UTFE:

```

/%F
ABSOLUT +010282BB=010282BB : C5D54040 EN
/%F
% AID0352 Kein weiterer Treffer im Suchbereich

```

Daten-Ausgabe mit %DISPLAY mit der impliziten UTF8-Interpretation:

```

/%D S8 %X
V'010282BF' = S8      + #'00000000'
010282BF (00000000) 4DC3BC6E 6368656E 2D444520 20D09C      München-DE ?
010282CE (0000000F) D18ED0BD D185D0B5 D0BD2D42 5920204D      юнхен-BY M
010282DE (0000001F) 756E6963 682D454E 2020                unich-EN

```

Verwendung des Wildcard-Symbols % in 'string', um nun im UTF8-Bereich zu suchen:

```
/%F '%E' IN S8
```

```
ABSOLUT +010282C8=010282C8 : 44452020 D09CD18E D0BDD185 DE юHX
```

```
/%F
```

```
ABSOLUT +010282E3=010282E3 : 2D454E20 20 -EN
```

```
/%AID CCS=UTF16
```

```
/%D S16 %X
```

```
V'0102830A' = S16 + #'00000000'
```

```
0102830A (00000000) 004D00FC 006E0063 00680065 006E003D München=
```

```
0102831A (00000010) 00440045 00200020 041C042E 041D0425 DE юHX
```

```
0102832A (00000020) 0415041D 003D0042 00590020 0020004D EH=BY M
```

```
0102833A (00000030) 0075006E 00690063 0068003D 0045004E unich=EN
```

```
0102834A (00000040) 00200020
```

Verwendung des Schlüsselworts ALL:

```
/%F ALL '%ch%' IN S16
```

```
ABSOLUT +0102830A=0102830A : 004D00FC 006E0063 00680065 Münche
```

```
ABSOLUT +0102833A=0102833A : 0075006E 00690063 0068003D unich=
```

```
/%AID CCS=ISO88591
```

```
/%F ALL 'M%n' IN C=FTEST@@@
```

```
FTEST@@@+000001BD=010011BD : 4D756E69 63682D45 4E000000 Munich-EN...
```

```
FTEST@@@+000001D8=010011D8 : 4DFC6E63 68656E2D 44450000 München-D..?
```

Verwendung der AID-Funktionen %C() und %UTF16():

```
/%F %C('M') IN SE
```

```
ABSOLUT +01028290=01028290 : D49EB7D5 C3C8C5D5 6DC4C540 M..NCHEN_DE
```

Wird %C() verwendet, dann interpretiert %FIND *find-bereich* als in demjenigen CCS codiert, der durch das Kommando %AID EBCDIC-Kommando festgelegt wurde (=EDF03IRV).

```
/%F %UTF16('M') IN S16
```

```
ABSOLUT +0102830A=0102830A : 004D00FC 006E0063 00680065 Münche
```

Wird %UTF16() verwendet, dann interpretiert %FIND *find-bereich* als UTF16-codiert.

### 9.1.2 Sedezimal-Literal

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

maximale Länge: 80 Sedezimal-Stellen (entspricht 40 Bytes).

Ein Literal mit ungerader Stellenzahl wird rechts mit X'0' ergänzt.

Zeichenvorrat für *f*: jedes Zeichen von 0 - 9 und A - F.

Die Typmodifikation %UTF16 ist für ein Sedezimal-Literal zulässig. Durch diese Typmodifikation wird das Literal wie ein Character-Literal behandelt (siehe [Abschnitt „Character-Literal“ auf Seite 113](#)).

%DISPLAY

Das Literal wird ausgegeben. Es kann über eine Typmodifikation konvertiert werden.

%FIND

*f* kann auch das Wildcard-Symbol '%' sein. Es steht jeweils für ein beliebiges Zeichen und wird von %FIND immer als Treffer gemeldet.

%MOVE

Das Literal wird linksbündig in der Länge des Literals in das Empfangsfeld übertragen. Ist das Literal länger als das Empfangsfeld, wird die Übertragung mit einer Meldung abgewiesen.

%SET

Das Literal wird linksbündig übertragen. Ist das Empfangsfeld länger als das Literal, wird rechts mit X'00' aufgefüllt. Ist das Literal länger als das Empfangsfeld, wird rechts abgeschnitten.

Dieses Literal kann für die Übertragung in ein Empfangsfeld mit beliebiger Datentyp-Definition verwendet werden.

### 9.1.3 Binär-Literal

{B'b...b' | 'b...b'B}

maximale Länge: 80 Binärstellen (entspricht 10 Bytes).

Es wird rechts auf Bytelänge (8 Binärstellen) mit binär Null B'0' ergänzt.

Zeichenvorrat für *b*: die Zeichen 0 und 1

**%DISPLAY**

Das Literal wird ausgegeben. Es kann über eine Typmodifikation konvertiert werden.

**%FIND**

B'b...b' kann nicht angegeben werden.

**%MOVE**

Das Literal wird linksbündig in der Länge des Literals in das Empfangsfeld übertragen. Ist das Literal länger als das Empfangsfeld, wird die Übertragung mit einer Meldung abgewiesen.

**%SET**

Das Literal wird linksbündig übertragen. Ist das Empfangsfeld länger als das Literal, wird rechts mit binär Null aufgefüllt. Ist das Literal länger als das Empfangsfeld, wird rechts abgeschnitten.

Dieses Literal kann in ein Empfangsfeld mit beliebiger Datentyp-Definition übertragen werden.



## 9.2 Numerische Literale

### 9.2.1 Ganzzahl

`[{±}]n`

maximale Länge: 20 Ziffern

Wertebereich:  $-10^{21} \leq n \leq +10^{20}-1$



Die interne Darstellung einer Ganzzahl ist für den Anwender undefiniert, das heißt, referenziert der Anwender in einem AID-Kommando die interne Darstellung, ist das Ergebnis undefiniert.

Beispiel: `%D 12345 %X / %M 123456789 INTO V'xxxx'`

Aus Kompatibilitätsgründen ist die interne Darstellung im Bereich von  $2^{31} \leq n \leq +2^{31}-1$  wie `%FL4`.

`%DISPLAY`

Das Literal wird ausgegeben. Es kann über eine Typmodifikation konvertiert werden.

`%FIND`

Eine Ganzzahl kann nicht angegeben werden.

`%MOVE`

Die Ganzzahl wird als sedezimaler Wert in einem Wort (4 Bytes) aufbereitet und linksbündig im Empfangsfeld abgelegt. Ist das Empfangsfeld zu kurz, wird die Übertragung mit einer Fehlermeldung abgewiesen.

`%SET`

Die Ganzzahl kann in jedes numerische Empfangsfeld übertragen werden; sie wird gegebenenfalls dem Typ des Empfangsfeldes angepasst und werterhaltend übertragen.

### 9.2.2 Sedezimalzahl

`#'x...x'`

maximale Länge: 16 Sedezimal-Stellen (entspricht dem Speichertyp `%FL8`, Ganzzahl mit Vorzeichen in Integer-Darstellung)

Wertebereich:

Bei einer max. Länge von 8 Sedezimalstellen:  $-2^{31} \leq \#'x...x' \leq +2^{31}-1$  (`%FL4`)

Bei einer Länge von mindestens 9 Sedezimalstellen:  $-2^{63} \leq \#'x...x' \leq +2^{63}-1$  (`%FL8`)

Zeichenvorrat für  $x$ :

Negative Sedezimalzahl:

32-Bit-Zahlen:

haben genau 8 Sedezimalziffern, wobei das erste Bit in der linkensten Ziffer das Vorzeichen beschreibt. Um einen negativen Wert zu beschreiben, muss diese Ziffer aus dem Bereich X'8' , X'9' , ..., X'F' kommen.

64-Bit-Zahlen:

haben genau 16 Sedezimalziffern; wie bei einer 32-Bit Sedezimalzahl muss die linkeste Ziffer aus dem Bereich X'8' , X'9' , ..., X'F' kommen.

Beispiel:

#'FFFFFFF' beschreibt eine 32-Bit-Sedezimalzahl mit Wert -1;

#'0FFFFFFF' beschreibt eine 64-Bit-Sedezimalzahl mit dem Wert +2\*\*32-1;

Damit bleibt das bisherige Verhalten bei 32-Bit-Sedezimalzahlen von älteren AID Versionen erhalten.

%DISPLAY

Das Literal wird ausgegeben. Es kann über eine Typmodifikation konvertiert werden.

%FIND

Eine Sedezimalzahl kann nicht angegeben werden.

%MOVE

Die Sedezimalzahl wird in Wortlänge (4 Bytes) aufbereitet und linksbündig im Empfangsfeld abgelegt. Ist das Empfangsfeld zu kurz, wird die Übertragung mit einer Fehlermeldung abgewiesen.

%SET

Die Sedezimalzahl kann in jedes numerische Empfangsfeld übertragen werden; sie wird gegebenenfalls dem Typ des Empfangsfeldes angepasst und werterhaltend übertragen.

### 9.2.3 Dezimalpunktzahl

[{±}]n.m

maximale Länge: 18 Ziffern, ein Dezimalpunkt und ein Vorzeichen.

Vor der höchstwertigen Stelle kann ein Vorzeichen stehen. Ein Dezimalpunkt kann an jeder Position innerhalb der Ziffernfolge angegeben werden. Soll er an höchster Stelle stehen, muss eine Null davor gesetzt werden.

%DISPLAY

Das Literal wird ausgegeben.

%FIND/%MOVE

Eine Dezimalpunktzahl kann nicht angegeben werden.

%SET

Die Dezimalpunktzahl kann in jedes numerische Empfangsfeld übertragen werden; sie wird gegebenenfalls dem Typ des Empfangsfeldes angepasst und werterhaltend übertragen.

## 9.2.4 Gleitpunktzahl

[{±}]mantisseE[{±}]exponent

Die Gleitpunktzahl wird intern mit doppelter Genauigkeit (8 Bytes) aufgebaut. Haben *mantisse* oder *exponent* kein Vorzeichen, werden sie positiv interpretiert. Innerhalb der Gleitpunktzahl dürfen keine Leerzeichen geschrieben werden.

*mantisse*

maximale Länge: 16 signifikante Ziffern, ein Dezimalpunkt und ein Vorzeichen  
*mantisse* muss einen Dezimalpunkt enthalten, der an jeder beliebigen Stelle innerhalb stehen kann. Soll er an der höchsten Stelle stehen, muss eine Null vorangehen.

*exponent*

maximale Länge: 2 Ziffern und ein Vorzeichen  
Wertebereich:  $-75 \leq \textit{exponent} \leq 76$ .

%DISPLAY

Das Literal wird ausgegeben.

%FIND/%MOVE

Eine Gleitpunktzahl kann nicht angegeben werden.

%SET

Die Gleitpunktzahl kann in jedes numerische Empfangsfeld übertragen werden; es wird entsprechend dem Typ des Empfangsfelds angepasst und werterhaltend übertragen.



---

## 10 Schlüsselwörter

Schlüsselwörter sind für AID festgelegte Vereinbarungen, die mit einem Prozent-Zeichen beginnen. Sie stehen für Speichertypen, Register, Befehlszähler, Speicherklassen, Systeminformationen, Durchlaufzähler, logische Werte, Vorschubsteuerung, Adressumschaltung, Ausgabe der aktuellen Aufrufhierarchie, Befehlstypen und Ereignisse.

In einer komplexen Speicherreferenz können Sie die Schlüsselwörter für Speicherklassen, Register, Befehlszähler und Durchlaufzähler verwenden. Die impliziten Speichertypen und Längen sind im jeweiligen Abschnitt angegeben.

### 10.1 Allgemeine Speichertypen

Mit den Schlüsselwörtern für Speichertypen können Sie eine Speicherstelle oder ein Literal anders als definiert interpretieren lassen (siehe [Abschnitt „Typmodifikation“ auf Seite 95](#)). Das kann in den folgenden Fällen notwendig oder sinnvoll sein:

Beim %SET sind die Speichertypen von *sender* und *empfänger* nicht miteinander verträglich.

Beim %DISPLAY wollen Sie sich eine Speicherstelle oder ein Literal konvertiert ausgeben lassen. Jedem Speichertyp ist implizit ein Ausgabetyt zugeordnet, der festlegt, wie der Speicherinhalt ausgegeben wird.

Bei der Adressierung wollen Sie eine Speicherstelle in die Rechnung mit einbeziehen.

Mit der wahlweisen Längenangabe *L-mod* können Sie zugleich die Länge modifizieren (siehe [Abschnitt „Längenmodifikation“ auf Seite 98](#)). Dies ist bei Literalen nicht zugelassen. Zwischen Typ und Längenangabe darf kein Leerzeichen geschrieben werden. Die Längenangabe kann jede Form der Längenmodifikation annehmen.

`%X[L-mod]`    Sedezimal, Länge  $1 \leq n \leq 65.535$   
Der Standard-Speichertyp für eine virtuelle Adresse ist %XL4.  
Ausgabetyt: Dump (Sedezimal und Character)

`%C[L-mod]`    Character, Länge  $1 \leq n \leq 65.535$   
Ausgabetyt: Character

`%UTF16[L-mod]`  
Unicode-Character, Länge  $2 \leq n \leq 65.534$ ,  
Die Länge muss ein Vielfaches von 2 sein.  
Ausgabetyt: Dump (Hexadezimal und Character)

|                 |                                                                                                                                                                                            |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| %P[L-mod]       | Gepackt, Länge $1 \leq n \leq 9$ , darf außer dem Vorzeichen (letztes Halbbyte) nur Ziffern enthalten.<br>Ausgabety: numerisch (Ganzzahl mit Vorzeichen)<br>Für Literale nicht zugelassen. |
| %D[L-mod]       | Gleitpunkt, Länge $n = 4, 8$ oder $16$ Bytes<br>Ausgabety: numerisch (Gleitpunktdarstellung)<br>Für Literale nicht zugelassen.                                                             |
| %F[L-mod]<br>%H | Binär mit Vorzeichen (Integer), Länge $n = 1..8$ Bytes<br>entspricht %FL2<br>Ausgabety: numerisch (Ganzzahl mit Vorzeichen)<br>%H ist für Literale nicht zugelassen.                       |
| %A[L-mod]<br>%Y | Adresse, Länge $n = 1..8$ Bytes<br>entspricht %AL2<br>Ausgabety: numerisch (Ganzzahl ohne Vorzeichen)<br>Für Literale nicht zugelassen.                                                    |

### 10.1.1 Speichertypen zur Invertierung der Byte-Reihenfolge eines Datenelements

Modifikationen des Typs %E dienen zur Invertierung der Byte-Reihenfolge eines Speicherinhalts.

Sie funktioniert mit:

|              |                                                                     |
|--------------|---------------------------------------------------------------------|
| %n           | Mehrzweckregister                                                   |
| %nG          | AID-Mehrzweckregister                                               |
| %MR          | alle 16 Mehrzweckregister in Tabellenform                           |
| %nQ          | Gleitpunktregister, $n = 0,4$                                       |
| %nD E        | Gleitpunktregister, $n = 0,2,4,6$                                   |
| %FR          | alle 4 Gleitpunktregister mit doppelter Genauigkeit in Tabellenform |
| %PC          | Befehlszähler (Program Counter)                                     |
| %PCB         | Process Control Block                                               |
| %nAR         | Zugriffsregister                                                    |
| field        | Daten vom geladenen Programm                                        |
| %TCB         |                                                                     |
| %CLASS6      | Länge muss kleiner als 1024 sein                                    |
| %CLASS6ABOVE | Länge muss kleiner als 1024 sein                                    |
| %CLASS6BELOW | Länge muss kleiner als 1024 sein                                    |
| %CLASS5      | Länge muss kleiner als 1024 sein                                    |
| V'xxxxxxxx'  | virtuelle Adresse                                                   |

Modifikationen des Typs %E können mit den Kommandos %MOVE, %SET, %DISPLAY verwendet werden.

Nach Anwendung der Modifikation %E auf einen Operanden hat dieser den hexadezimalen Speichertyp (%X), d.h. Ausgabetyt Dump.

### Beispiele

#### 1. %E invertiert den Speicherinhalt des Register

```
/%DISPLAY %4
%4                = 01001444
```

Mit %E:

```
/%DISPLAY %4%E
%4                = 44140001
```

#### 2. %E invertiert den Speicherinhalt von Adresse V'00000066'

```
/%DISPLAY V'00000066'
V'00000066' = SEND2 + #'00000066'
00000066 (00000066) 11223344
```

Mit %E:

```
/%DISPLAY V'00000066'%E
V'00000066' = SEND2 + #'00000066'
00000066 (00000066) 44332211
```

Sie können %E zwar mit der Längenmodifikation %Eln verwenden, allerdings darf die Länge 1024 nicht überschreiten. Das Schlüsselwort %CLASSx%E muss mit der Längenmodifikation verwendet werden, und die Länge muss kleiner als 1024 sein.

### Beispiele

```
1. /%DISPLAY %5
%5                = 12345678
```

```
/%MOVE %5%E INTO %10
/%DISPLAY %10
%10               = 78563412
```

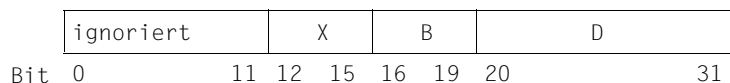
```
2. /%DISPLAY %5
%5                = 12345678
```

```
/%SET %5%EL2 INTO %10
%10               = 34120000
```

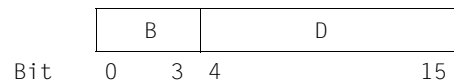
## 10.2 Speichertypen zur Interpretation von Maschinenbefehlen

Diese Speichertypen werden benutzt, um eine Adresse zu errechnen, die in den Hauptspeicher-Operanden von Maschinenbefehlen durch Basisregister und Distanz oder Index-, Basisregister und Distanz hinterlegt ist. Erst der nachfolgende Pointer-Operator "->" veranlasst die Errechnung der Adresse. Ohne nachfolgenden Pointer-Operator wirkt %SX wie %XL4 und %S wie %XL2. Beispiele dazu finden Sie im [Abschnitt „Indirekte Adressierung“-> / "\\*\\*\\*\\*" auf Seite 91.](#)

%SX SX-Adresse, Länge 4 Bytes, Index-Basis-Distanz (X-B-D), entspricht einem Maschinenbefehl im SX-Format  
Die Indexregister-Nummer X wird nur ausgewertet, wenn sie  $\neq 0$  ist.  
Ausgabety: Sedezimal



%S S-Adresse, Länge 2 Bytes Basis-Distanz (B-D)  
Die Basisregister-Nummer B wird nur ausgewertet, wenn sie  $\neq 0$  ist.  
Ausgabety: Sedezimal





## 10.3 Programmregister und Befehlszähler

Die Programmregister (Mehrzweckregister und Gleitpunktregister) und der Befehlszähler (Program Counter) werden von AID über Schlüsselwörter angesprochen. Sie können den Inhalt anzeigen lassen (%DISPLAY), verändern (%MOVE, %SET) oder zur Adressierung benutzen. Die Programmregister liegen im privilegierten Bereich und können über ihre virtuellen Adressen nicht angesprochen werden. Die Gleitpunktregister belegen einen gemeinsamen Speicherplatz:

%0Q überdeckt %0D und %2D; %4Q überdeckt %4D und %6D.

Der Inhalt von Mehrzweckregistern wird in Subkommandobedingungen und arithmetischen Ausdrücken von AID als numerischer Wert mit Vorzeichen behandelt, wie es seinem Typ %FL4 entspricht.

Vor einem Pointer-Operator hingegen nimmt AID den Typ %AL4 an und der Registerinhalt kann ohne Typmodifikation als Adresse verwendet werden. Ausgegeben wird ein Register als Sedezimalzahl. Um den numerischen Wert mit Vorzeichen ausgeben zu lassen, müssen Sie für die Ausgabe eine Typmodifikation mit %F vornehmen.

In ASSEMBH haben die Programmregister auch symbolische Namen `_Rn`, die jedoch nur beim symbolischen Testen von Assemblerprogrammen verwendet werden können.

```
%PC  Befehlszähler (Program Counter), Typ %AL4
%n   Mehrzweckregister 0 ≤ n ≤ 15, Speichertyp %FL4,
     Ausgabe als Sedezimalzahl
%nE  Gleitpunktregister mit einfacher Genauigkeit n = {0,2,4,6}, Typ %DL4
%nD  Gleitpunktregister mit doppelter Genauigkeit n = {0,2,4,6}, Typ %DL8
%nQ  Gleitpunktregister mit vierfacher Genauigkeit n = {0,4}, Typ %DL16

%MR  alle 16 Mehrzweckregister in Tabellenform aufbereitet
%FR  alle 4 Gleitpunktregister mit doppelter Genauigkeit
     in Tabellenform aufbereitet

%nAR  Zugriffsregister 0 ≤ n ≤ 15, Speichertyp %FL4, Ausgabe als
     Sedezimalzahl (AR-Modus)
%AR  alle 16 Zugriffsregister in Tabellenform aufbereitet
     (AR-Modus)
```

Das Schlüsselwort %PC und die Register können Sie bei Bedarf mit einem Index versehen. Diesen Index brauchen Sie nur, wenn Sie ein Programm bearbeiten, das Contingency-Prozesse definiert hat bzw. in der STXIT-Verarbeitung von AID unterbrochen wurde. Wenn Sie nicht die Informationen zum unterbrochenen Contingency- bzw. STXIT-Prozess, sondern zu einer anderen Prozess-Ebene, z.B. dem Basis-Prozess, wollen, dann müssen Sie den entsprechenden Index angeben. Diesen Index können Sie mit %DISPLAY %PCBLST feststellen (siehe „[Makroaufrufe an den Ablaufteil](#)“ [10]).

Der Index wird folgendermaßen angegeben: schlüsselwort(index).

## 10.4 AID-Register

Die AID-Register liegen in dem Speicherbereich, der für AID reserviert ist, damit können die AID-Register von jedem AID-Arbeitsbereich aus angesprochen werden, ohne die Arbeitsbereich-Vereinbarung zu verändern. Die AID-Register entsprechen in Typ und Länge den Programmregistern.

Schlüsselwörter zum Ansprechen aller AID-Register gibt es nicht.

```
%nG      AID-Mehrzweckregister  $0 \leq n \leq 15$ , Speichertyp %FL4,
          Ausgabe als Sedezimalzahl
%nGD     AID-Gleitpunktregister mit doppelter Genauigkeit  $n = \{0,2,4,6\}$ ,
          Typ %DL8
```

## 10.5 Speicherklassen

Mit den folgenden Schlüsselwörtern können Sie Speicherbereiche ansprechen. Sie können als Bereich bei %CONTROL $n$ , %DISASSEMBLE, %DISPLAY, %FIND und %TRACE angegeben werden. In einer komplexen Speicherreferenz können Sie die Schlüsselwörter für Speicherklassen mit allen Attributen verwenden:

Name, Adresse, Inhalt, Länge, Typ.

Im Klasse-5-Speicher belegt Ihr Programm privilegierte und nicht-privilegierte Bereiche. Auf beide Bereiche können Sie mit dem Schlüsselwort %CLASS5 zugreifen, auf den privilegierten Bereich jedoch nur mit höherer Test-Privilegierung.

Mit den Schlüsselwörtern %CLASS5 und %CLASS5BELOW wird derselbe Adressraum bezeichnet. Entsprechendes gilt für %CLASS6 und %CLASS6BELOW.

```
%CLASS5      Klasse-5-Speicher, Typ %X
%CLASS6      Klasse-6-Speicher, Typ %X
```

Zusätzlich gibt es beim Testen von Programmen oder beim Bearbeiten von Speicherabzügen noch die folgenden Schlüsselwörter, ebenfalls alle vom Typ %X:

```
%CLASS5BELOW Klasse-5-Speicher, unterhalb der 16MB-Grenze ( $\neq$ %CLASS5)
%CLASS5ABOVE  Klasse-5-Speicher, oberhalb der 16MB-Grenze
%CLASS6BELOW Klasse-6-Speicher, unterhalb der 16MB-Grenze ( $\neq$ %CLASS6)
%CLASS6ABOVE  Klasse-6-Speicher, oberhalb der 16MB-Grenze
```

## 10.6 Systeminformationen

Über die Schlüsselwörter für Systeminformationen erhalten Sie mit %DISPLAY die entsprechenden Informationen über eine Task. Wird mehr als nur ein Wert für das entsprechende Schlüsselwort zurückgeliefert, führt AID eine Aufbereitung durch und gibt eine Tabelle aus.

|                      |                                                                                                    |
|----------------------|----------------------------------------------------------------------------------------------------|
| %CC                  | Condition Code                                                                                     |
| %PCB                 | Process Control Block (kann wie %PC indiziert werden)                                              |
| %PCBLST              | Liste aller Process Control Blocks                                                                 |
| %LINK                | Name des zuletzt nachgeladenen Segments, das mit %ON %LPOV festgestellt wurde                      |
| %PM                  | Program Mask                                                                                       |
| %AUD1                | Hardware-Audit-Tabelle beginnend mit dem jüngsten Eintrag; nur wenn bei Systemgenerierung angelegt |
| %AMODE               | Systeminformationsfeld zum Adressierungsmodus. Kann nur mit %MODE24 oder %MODE31 verändert werden  |
| %ASC                 | ASC-Modus (bzgl. AR-Modus bedeutet: X'00' = aus; X'01' = ein)                                      |
| %DS[(ALET/SPID-qua)] | Information über SPIDs und/oder ALETs der aktiven Datenräume                                       |
| %LOC(speicherref)    | maschinennahe Lokalisierungsinformation zu einer Adresse im ausführbaren Teil                      |
| %HLLOC(speicherref)  | symbolische Lokalisierungsinformation zu einer Adresse im ausführbaren Teil                        |
| %SORTEDMAP           | Liste aller CSECTs und COMMONs des Benutzerprogramms (namen- und adresssortiert)                   |

```
%MAP [ { { CTX=kontext [•L=ladeeinheit]
           L=ladeeinheit
           SCOPE = { USER
                   }
           } } ]
```

{CTX=kontext | L=ladeeinheit}

Bei Angabe eines Pfads werden alle CSECTs/Commons des angegebenen Kontextes oder der angegebenen Ladeeinheit aufgelistet.

SCOPE=USER

CSECTs/Commons der Standard-Kontexte CTXPHASE bzw. LOCAL#DEFAULT und der durch den BIND-Makro mit Operand LNKCTX[@] gebildeten Kontexte werden aufgelistet.

SCOPE=ALL

Zusätzlich zu den CSECTs/Commons der benutzerdefinierten Kontexte wird die MAP aller Kontexte ausgegeben, an die sich das Programm konnektiert hat wie z.B. DSSM-Subsysteme oder Userpool-Kontexte.

Alle BLS-Namen (Kontext, Ladeeinheit, CSECT und COMMON) werden ungekürzt ausgegeben. Innerhalb der Kontexte und Ladeeinheiten wird die ausgegebene Liste nach CSECT-Namen sortiert.

## Beispiele

1. /%D %HLLOC(PROG=UPRONUM.S'22DIS'->)

### AID-Ausgabe

```
V'000083EC' = CONTEXT : LOCAL#DEFAULT
              SMOD    : UPRONUM
              PROC    : UPRONUM
              PARAGRAPH: ENTPA
              SRC-REF  : 22DIS
              LABEL   : ENTPA
```

2. /%D %LOC(PROG=UPRONUM.S'22DIS'->)

### AID-Ausgabe

```
V'000083EC' = CONTEXT : LOCAL#DEFAULT
              LMOD    : %UNIT
              SMOD    : UPRONUM
              OMOD    : UPRONUM
              CSECT   : UPRONUM (00008018) + 000003D4
```

3. /%D %MAP

### AID-Ausgabe

```
*** TID: 00230056 *** TSN: 0FZB *****
CURRENT PC: 00002000 CSECT: LLMTEST2 *****
**CSECT-LISTING(MAP) OF CONTEXT : LOCAL#DEFAULT
**MAP OF LOAD UNIT : %UNIT
CSECT-NAME          START      SIZE  VER/DATE_OF_MOD
ASSTEST             00000150 000830 .....
ASS2                 00000980 000008 .....
COMM1                00001000 00012C %COMMON.....
LLMTEST1            00000000 000150 .....
**CSECT-LISTING(MAP) OF CONTEXT : CTX2
**MAP OF LOAD UNIT : LLMTEST2
CSECT-NAME          START      SIZE  VER/DATE_OF_MOD
ASSTEST             00002078 000650 .....
ASS2                 000026C8 000008 .....
COMM1                FFFFFFFF 000000 %COMMON.....
LLMTEST2            00002000 000078 .....
```

## 10.7 Durchlaufzähler

Der Durchlaufzähler wird zu jedem Subkommando angelegt. In ihm wird gezählt, wie oft das Subkommando durchlaufen wurde. Innerhalb des Subkommandos kann der eigene Zähler immer mit %• angesprochen werden. Erhält das Subkommando einen Namen, so bekommt auch der Zähler einen Namen. In diesem Fall kann der Zähler außerhalb des Subkommandos mit %•subkdoname angesprochen werden. Die Ausführung eines Subkommandos können Sie vom Stand des Durchlaufzählers %•subkdoname abhängig machen, indem Sie ihn in einer Bedingung abfragen (siehe [Kapitel „Subkommando“ auf Seite 51](#)).

Die Zuweisung eines numerischen Wertes kann mit %SET erfolgen. Der Inhalt kann mit %DISPLAY gelesen werden. Der Zähler wird erhöht, wenn das zugehörige Subkommando durchlaufen wird. Den Durchlaufzähler können Sie überall dort verwenden, wo ein numerischer Wert zulässig ist.

```
%•[subkdoname] Variable vom Typ %FL4  
                 subkdoname ist der Name des zugehörigen Subkommandos.  
                 Mit der Kurzform %• bezeichnen Sie den Durchlaufzähler des  
                 gerade aktiven Subkommandos.
```

## 10.8 Logische Werte

Diese beiden Schlüsselwörter können zur Wertzuweisung (%SET) in logische Variablen aus Fortran-Programmen verwendet werden.

```
%TRUE  
%FALSE
```

## 10.9 Vorschubsteuerung

Die beiden Schlüsselwörter zur Vorschubsteuerung wirken nur auf das Ausgabemedium SYSLST. Sie können nur bei %DISPLAY angegeben werden.

```
%NP           Beginn einer neuen Seite  
%NL[(n)]     Ausgabe von n Leerzeilen,  $1 \leq n \leq 255$   
              Standardwert für n ist 1.
```

## 10.10 Adressumschaltung

Bei der XS-Programmierung werden 31-Bit-Adressen verwendet statt der sonst üblichen 24-Bit-Adressen. Mit diesen beiden Schlüsselwörtern können Sie den Adressierungsmodus des Testobjekts oder die Adressinterpretation bei indirekter Adressierung verändern.

`%MOVE %MODE31 INTO %AMODE` verändert den Adressierungsmodus

`%AINT %MODE24` verändert die Adressinterpretation in AID bei indirekter Adressierung

`%MODE24` 24-Bit-Adressierung

`%MODE31` 31-Bit-Adressierung

## 10.11 Aktuelle Aufrufhierarchie

Bei `%SDUMP` veranlasst das Schlüsselwort `%NEST` die Ausgabe der aktuellen Aufrufhierarchie.

`%NEST` Ausgabe der aktuellen Aufrufhierarchie

## 10.12 Kriterium bei `%CONTROLn` und `%TRACE`

Mit diesen Schlüsselwörtern werden Befehle oder Anweisungen aus den Programmiersprachen in Typ-Gruppen eingeteilt. Die Schlüsselwörter können in den AID-Kommandos `%CONTROLn` und `%TRACE` als Überwachungskriterium (Operand *kriterium*) angegeben werden.

Bei `%CONTROLn` wird das zugehörige Subkommando bearbeitet, wenn ein Befehl oder eine Anweisung der zu überwachenden Gruppe zur Ausführung ansteht.

Bei `%TRACE` wird eine Protokollzeile ausgegeben, wenn ein Befehl oder eine Anweisung der zu überwachenden Befehlsgruppe ausgeführt wird. Beim symbolischen Test findet die Ausgabe statt, bevor die entsprechende Anweisung ausgeführt wird, beim maschinennahen Testen wird protokolliert, nachdem der Befehl ausgeführt wurde.

Standardwert ist das symbolische *kriterium* `%STMT`. Das hat zur Folge, dass bei einem `%CONTROLn` oder `%TRACE` mit einer Bereichsangabe auf Maschinencode-Ebene immer ein Schlüsselwort für *kriterium* angegeben werden muss, falls nicht noch aus einem vorhergehenden `%CONTROLn` bzw. `%TRACE` ein Überwachungskriterium gültig ist.

| <i>kriterium</i>                     | <b>Befehls- oder Anweisungsgruppe</b>                                                                                  |
|--------------------------------------|------------------------------------------------------------------------------------------------------------------------|
| Beim Testen auf Maschinencode-Ebene: |                                                                                                                        |
| %INSTR                               | alle Maschinenbefehle, die durchlaufen werden                                                                          |
| %B                                   | Verzweigungsbefehle (das sind die Maschinenbefehle BAL, BALR, BAS, BASSM, BASR, BC, BCR, BCT, BCTR, BSM, BXH und BXLE) |
| %BAL                                 | Unterprogrammaufrufe (durch die Maschinenbefehle BAL, BALR, BAS, BASSM und BASR).                                      |
| Beim Testen auf symbolischer Ebene:  |                                                                                                                        |
| %STMT                                | alle Anweisungen, die durchlaufen werden                                                                               |
| %ASSGN                               | Zuweisungs-Anweisungen                                                                                                 |
| %CALL                                | SUBROUTINE-Aufrufe (CALL-Anweisungen)                                                                                  |
| %COND                                | IF(...) THEN-, ELSE IF(...) THEN-, ELSE- und IF(...) -Anweisungen                                                      |
| %DB                                  | Anweisungen zum Aufruf einer Datenbank                                                                                 |
| %EXCEPTION                           | bedingte Anweisungszweige                                                                                              |
| %GOTO                                | GOTO-Anweisungen                                                                                                       |
| %IO                                  | Ein-/Ausgabe-Anweisungen                                                                                               |
| %LAB                                 | Anweisungen nach Label                                                                                                 |
| %PROC                                | STOP-, END-, RETURN-, SUBROUTINE- und FUNCTION-Anweisungen                                                             |
| %SORT                                | MERGE- und SORT-Anweisungen                                                                                            |

## 10.13 Ereignis bei %ON

Diese Schlüsselwörter stehen für die Schreibüberwachung, Programmfehler, Programmbeendigungen, Supervisor-Calls und andere Ereignisse während des Programmablaufs. Sie können in dem Kommando %ON (Operand *ereignis*) angegeben werden. Der Operand *ereignis* legt fest, bei welchem Vorfall das Programm unterbrochen werden soll, um das zugehörige Subkommando zu bearbeiten.

Wenn mehrere %ON-Kommandos mit unterschiedlichen *ereignis*-Vereinbarungen gleichzeitig aktiv sind und auch zutreffen, führt AID die zugehörigen Subkommandos in der Reihenfolge aus, in der die Schlüsselwörter in der folgenden Tabelle aufgeführt sind. Treffen verschiedene %TERM-Ereignisse zu, werden die zugehörigen Subkommandos nach dem FIFO-Prinzip abgearbeitet.

Die Schreibüberwachung kann nicht gleichzeitig für verschiedene Bereiche angemeldet sein. Ein neuer %ON %WRITE(...) überschreibt einen früher eingegebenen (siehe auch [Abschnitt „%ON %WRITE mit %INSERT, %CONTROLn und %TRACE“ auf Seite 141](#)).

Zur Auswahl des geeigneten %TERM finden Sie weitere Informationen im Handbuch „[Makroaufrufe an den Ablaufteil](#)“ [10].

| Ereignis             | Subkommando wird bearbeitet:                                                                                                                                                                                       |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| %WRITE(speicherref)  | nach Überschreiben des mit speicherref bezeichneten Speicherbereichs.                                                                                                                                              |
| %ERRFLG(z)           | nach Auftreten eines Fehlers mit dem angegebenen Ereigniscode und<br>vor Abbruch des Programms.                                                                                                                    |
| %INSTCHK             | nach Auftreten eines Adressierungsfehlers, eines ungültigen Systemaufrufs (SVC), nicht decodierbaren Operations-Codes, Seitenwechsel-Fehlers oder einer privilegierten Operation und<br>vor Abbruch des Programms. |
| %ARTHCHK             | nach Auftreten eines Datenfehlers, Divisionsfehlers, Exponenten-Überlaufs oder einer Mantisse Null und<br>vor dem Abbruch des Programms.                                                                           |
| %ABNORM              | nach Auftreten eines der Fehler, die mit den vorher beschriebenen Ereignissen erfasst werden, sowie eines DMS-Errors oder eines %ILLSTX.                                                                           |
| %ERRFLG              | nach Auftreten eines Fehlers mit beliebigem Fehlergewicht.                                                                                                                                                         |
| %SVC(z)<br>%SVC      | vor Ausführung des Systemaufrufs (SVC) mit der angegebenen Nummer.<br>vor Ausführung eines beliebigen Systemaufrufs (SVC).                                                                                         |
| %LPOV(name)<br>%LPOV | nach dem Laden des Segmentes mit dem angegebenen Namen.<br>nach dem Laden eines beliebigen Segmentes.                                                                                                              |



| Ereignis          | Subkommando wird bearbeitet:                                                                                                                                                                                         |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| %TERM(N[ORMAL])   | vor Programmbeendigung mit<br>TERM MODE = NORMAL, TERM oder TERMD.                                                                                                                                                   |
| %TERM(A[BNORMAL]) | vor Programmbeendigung mit<br>TERM MODE = ABNORMAL, TERMJ oder TRMJD.                                                                                                                                                |
| %TERM(D[U]MP)     | vor Programmbeendigung mit<br>TERM DUMP = Y, TERMD oder TRMJD.                                                                                                                                                       |
| %TERM(ND NODUMP)  | vor Programmbeendigung mit<br>TERM DUMP = NO, TERM oder TERMJ.                                                                                                                                                       |
| %TERM(P[RGR])     | vor Programmbeendigung mit<br>TERM UNIT = PRGR, TERM oder TERMD.                                                                                                                                                     |
| %TERM(S[TEP])     | vor Programmbeendigung mit<br>TERM UNIT = STEP, TERMJ oder TRMJD.                                                                                                                                                    |
| %TERM             | vor Programmbeendigung mit<br>TERM, TERMD, TERMJ oder TRMJD.                                                                                                                                                         |
| %ANY              | vor der Beendigung des Programms auf Grund eines<br>Programmfehlers bzw. durch einen TERM mit<br>beliebigen Operandenwerten oder TERMJ,<br>TERMD oder TRMJD, oder auf Grund eines DMS-<br>Errors oder eines %ILLSTX. |
| %ILLSTX           | vor Auftreten eines STXIT-Aufrufs während der<br>Abarbeitung eines vorangegangenen STXIT-<br>Aufrufs (STXIT im STXIT)                                                                                                |

$z$  ist eine Ganzzahl mit:  $1 \leq z \leq 255$ .  $z$  kann als maximal dreistellige vorzeichenlose Dezimalzahl oder als zweistellige Sedezimalzahl (#'ff') angegeben werden.

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



---

# 11 Spezielle Anwendungen

## 11.1 %ON und STXIT

Es gibt verschiedene Möglichkeiten, auf Ereignisse zu reagieren, die während des Programmablaufs auftreten:

- Einzelnen Ereignissen können im Programm STXIT-Routinen zugeordnet werden, die beim Eintreten solcher Ereignisse zu deren Bearbeitung durchlaufen werden (siehe „[Makroaufrufe an den Ablaufteil](#)“ [10]).
- Beim Testen mit AID können Ereignisse über das Kommando %ON angemeldet werden. Tritt ein solches Ereignis ein, wird das im %ON-Kommando angegebene Subkommando bearbeitet.

Ereignisse, für die im Programm STXIT-Routinen angemeldet wurden, können von AID nicht bearbeitet werden: AID bekommt vom Eintreten solcher Ereignisse keine Kenntnis. Für solche Ereignisse werden im %ON-Kommando angegebene Subkommandos deshalb nicht durchlaufen.

STXIT-Routinen werden u.a. von den Laufzeitsystemen der Compiler, von ILCS, von openUTM und den Datenbanksystemen angemeldet, z.B. für die Ereignisse "Programmfehler" oder "nicht behebbarer Programmfehler". Diese STXIT-Ereignisse entsprechen den Ereignissen %ERRFLG(zzz), %ERRFLG, %INSTCK, %ARTHCHK und %ABNORM im %ON-Kommando. Diese Ereignisse können mit AID nur dann bearbeitet werden, wenn das Anmelden der STXIT-Routinen unterdrückt wurde.

Für FOR1-Programme ohne Standard-Linkage wird das Anmelden von STXIT-Routinen durch Angabe der Option RUNOPT STXIT=NO unterdrückt. Bei COBOL-Programmen und Programmen, die mit Standard-Linkage arbeiten (C ab V2.0A, C++ ab V2.1A, COBOL85 ab V1.1A, COBOL2000 ab V1.0A, FOR1 ab V2.2A und PLI1 ab V4.1A) kann das Anmelden von STXIT-Ereignissen nicht verhindert werden. Jedoch bietet ILCS bei Fehlern, die den Speicher nicht verändern wie Adressenfehler oder unzulässiger Operationscode das folgende Verfahren an: nach der Bearbeitung der STXIT-Routinen stellt ILCS den alten Befehlszählerstand wieder her, reproduziert den Fehler und übergibt die Kontrolle an das System, so dass es anschließend möglich ist, den Fehler mit dem %ON-Kommando zu bearbeiten. Bei allen anderen Fehlern bricht ILCS das Programm ab. Über %ON %ANY oder %ON %TERM können Sie jedoch den Programmablauf vor dem Entladen stoppen und die Fehlerursache mit AID-Kommandos untersuchen.

Bei openUTM-Anwendungen können die UTM-STXIT-Routinen (für UTM-T und UTM-P im Dialog) durch Angabe der Option STXIT=OFF im START-Parameter ausgeschaltet werden. Falls openUTM unter ILCS läuft (Operand PROGRAM COMP=ILCS in der KDCDEF-Anweisung), dann bleiben jedoch die ILCS-STXIT-Routinen auch nach dem Abschalten der UTM-STXIT wirksam.

In Assembler- und C++/C - Programmen können eigene Routinen (bei C++/C: Signalbehandlung über Bibliotheksfunktion signal(), bei Assembler: Makro STXIT) zur Fehlerbehandlung geschrieben werden. Auch in diesem Fall ist es für AID nicht möglich, über das %ON-Kommando auf einen durch selbst programmierte Routinen abgefangenen Fehler zu reagieren. Allerdings können Sie mit %INSERT einen Testpunkt in die Fehlerbehandlungsroutinen setzen, dessen zugehöriges Subkommando im Fehlerfall ausgeführt wird.

## 11.2 Programme mit Überlagerungsstruktur

Standardmäßig geht AID davon aus, dass ein Programm ohne Überlagerungsstruktur gebunden ist. Es benutzt die einmal geladenen LSD-Sätze, ohne jedes Mal zu prüfen, ob die angesprochene CSECT in einem mittlerweile nachgeladenen Segment liegt. Falls Sie jedoch ein Programm testen, das statisch als Overlay gebunden wurde oder das Segmente dynamisch mit den Makroaufrufen BIND/UNBIND nach- oder entlädt, müssen Sie im Kommando %AID den Operanden OV=YES angeben, um sicherzustellen, dass AID bei jedem Zugriff auf die LSD überprüft, ob zwischenzeitlich nachgeladen wurde.

Bei Programmen mit Überlagerungsstruktur können Sie einen Testpunkt nur in ein Segment setzen, das zum Zeitpunkt der Kommandoeingabe geladen ist. Ebenso können Sie einen Testpunkt nur löschen, wenn das zugehörige Segment geladen ist. Wird das Segment ent- oder überladen, bleibt der Testpunkt dennoch erhalten, falls er nicht zuvor explizit mit %REMOVE gelöscht wurde. Wird das Segment erneut geladen, so ist auch der Testpunkt wieder gesetzt.

---

## 12 Einschränkungen und Wechselwirkungen

### 12.1 %ON %WRITE mit %INSERT, %CONTROL $n$ und %TRACE

Zwischen %ON *write-ereignis* und den AID-Kommandos %CONTROL $n$ , %INSERT und %TRACE sind die folgenden Wechselwirkungen zu beachten:

- Wenn ein %CONTROL $n$  oder ein %TRACE mit maschinennahem *kriterium* angemeldet ist, wird die Eingabe von %ON *write-ereignis* mit einer Fehlermeldung abgewiesen und umgekehrt.
- Wenn ein Maschinenbefehl durch einen %CONTROL $n$  oder %TRACE mit symbolischem *kriterium* mit der AID-internen Markierung (X'0A81') überschrieben wurde, bemerkt AID den schreibenden Zugriff dieses Befehls nicht.
- Wenn ein Maschinenbefehl durch den mit %INSERT vereinbarten Testpunkt mit der AID-internen Markierung überschrieben wurde, erkennt AID auch hier den schreibenden Zugriff dieses Befehls nicht.

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

## 12.2 Wechselwirkungen zwischen Ablaufüberwachung und Ausgabe oder Modifikation von Speicherinhalten

Wenn Sie mit %INSERT einen Testpunkt setzen, überschreibt AID den Befehlscode an der Adresse des Testpunkts mit einem SVC X'81'. Ebenso markiert AID den ersten Befehl der ausführbaren Anweisungen in *control-* und *trace-bereich* bei symbolischem *kriterium* mit X'0A81'. Diese Markierungen können Sie sich ansehen, wenn Sie mit %DISPLAY den betreffenden Befehlscode ausgeben lassen. Wenn Sie hingegen den Code mit %DISASSEMBLE rückübersetzen, tauscht AID den eingetragenen SVC gegen den Originalbefehl aus und Sie sehen den Befehlscode, so wie ihn der Compiler erzeugt hat. Ebenfalls greift AID auf den Originalcode zurück, wenn aus dem entsprechenden Befehl über die Typmodifikation %S oder %SX eine Adresse berechnet werden soll, die dann über Pointer (->) angesprochen wird.

Mit dem %FIND-Kommando können Sie die gesetzten Markierungen im Befehlscode suchen, indem Sie X'0A81' als Suchbegriff angeben.

Bei den Kommandos %MOVE und %SET werden eventuell eingetragene Markierungen nicht ersetzt. Das kann dazu führen, dass beim Übertragen von Befehlscode Markierungen verschwinden bzw. neue gesetzt werden, die AID mit seiner internen Kommandoverwaltung nicht mehr in Verbindung bringen kann. Sie müssen also selbst sicherstellen, dass vor dem Übertragen bzw. Überschreiben im betroffenen Befehlscode keine von AID eingetragenen SVCs enthalten sind. Testpunkte und Markierungen des %CONTROL<sub>n</sub> löschen Sie mit dem Kommando %REMOVE. Ein eventuell noch eingetragener %TRACE wird gelöscht, indem Sie das Programm mit %RESUME starten oder %TRACE 1 %INSTR eingeben.

## 12.3 Testpunkte in Common Memory Pools

Wird ein Testpunkt in einen Common Memory Pool gesetzt, so ist dieser Testpunkt nur der Task bekannt, die den Testpunkt gesetzt hat. Alle weiteren Tasks, die ebenfalls an den Common Memory Pool angeschlossen sind, bleiben an diesem Testpunkt hängen.

Ein weiterer Benutzer des Common Memory Pools müsste von Hand den Originalcode wieder einsetzen. Andernfalls muss die hängengebliebene Task warten, bis der Testpunkt durch die Task, die ihn eingetragen hat, explizit mit %REMOVE zurückgesetzt wird oder bis die Task sich beendet. Dadurch werden die von dieser Task gesetzten Testpunkte implizit entfernt.

Ein weiteres Problem entsteht, wenn die Verbindung von der Task, die den Testpunkt gesetzt hat, zum Common Memory Pool getrennt wird, ohne dass der Testpunkt mit %REMOVE entfernt wurde. Wenn die Task so beendet wird, unterbleibt die implizite Rücksetzung des Testpunkts, da zwischen der Task und dem Common Memory Pool keine Verbindung mehr besteht. Der Originalcode kann dann nur noch von Hand wiederhergestellt werden.

Um die geschilderten Probleme zu vermeiden, sollte der Code zum Testen lokal ablaufen, also nicht in einem Common Memory Pool. Falls aber ein Testpunkt in einem Common Memory Pool benötigt wird, sollte dieser unmittelbar nach Erreichen mit %REMOVE entfernt werden. Zumindest sollte während des Testens der Verbindungsabbau zum Common Memory unterdrückt werden, damit die Testpunkte implizit entfernt werden können.

## 12.4 Low-Level-Trace und -Control in Verbindung mit Contingencies

### 12.4.1 %TRACE

Bei Tasks mit ereignisgesteuerter Verarbeitung kann es beim Ablauf von %TRACE auf Maschinencode-Ebene zur fehlerhaften Protokollierung einzelner Maschinenbefehle kommen. Wenn aufgrund eines asynchronen Ereignisses eine Contingency-Routine aufgerufen wird und gleichzeitig ein Low-Level-Trace abläuft, können beim Eintritt in die Contingency-Routine Fehler auftreten. Dies gilt auch bei der Rückkehr in die von dem Ereignis unterbrochene Task.

Abhängig von der jeweiligen Programm- und Test-Konstellation kann an der Schnittstelle Basisprozess/Contingency folgendes Fehlverhalten auftreten:

- ein Befehl wird falsch protokolliert, z.B. mit falschen Registerinhalten
- ein Befehl geht verloren
- ein Befehl wird doppelt angezeigt

### 12.4.2 %CONTROL

Analog kann es bei der Befehlsüberwachung mit %CONTROL vorkommen, dass ein Subkommando ausgelassen oder zu häufig ausgeführt wird.



STXIT-Routinen, die bei Programmfehlern gestartet werden, sind von dem oben geschilderten Fehlverhalten nicht betroffen. Hier werden bei %TRACE und %CONTROL alle Befehle richtig und vollständig abgearbeitet.



---

# 13 Anhang

## 13.1 In Kommandofolgen und Subkommandos unzulässige SDF- und ISP-Kommandos

Liste der BS2000-Kommandos, die nicht in Kommandofolgen und Subkommandos stehen dürfen.

| <b>Kommando</b>        | <b>Kurzbeschreibung</b>                                                          |
|------------------------|----------------------------------------------------------------------------------|
| ABORT                  | Prozedur abbrechen                                                               |
| ADD-CJC-ACTION         | bedingte Ausführung einer Kommandofolge vereinbaren                              |
| BEGIN-PROCEDURE        | Prozedurdateiattribute festlegen                                                 |
| BREAK                  | Kommando-Modus anfordern                                                         |
| CATEGORY               | Lastverteilung einer Anlage steuern                                              |
| CANCEL-PROCEDURE       | Prozedur(ablauf) abbrechen                                                       |
| CHANGE-ACCOUNTING-FILE | aktuelle Abrechnungsdatei schließen, neue Datei anlegen                          |
| CHANGE-CONSLOG         | aktuelle Protokolldatei schließen, neue Datei anlegen                            |
| CHANGE-SERSLOG         | aktuelle SERSLOG-Datei schließen, neue Datei anlegen                             |
| DELON                  | ON-Kommando löschen                                                              |
| END-CJC-ACTION         | CJC-Kommandofolge abschließen                                                    |
| ENDON                  | beendet ON-Anweisungsfolge                                                       |
| ENDP                   | beendet Prozedurdatei                                                            |
| END-PROCEDURE          | beendet Prozedurdatei                                                            |
| EOF                    | Dateiende für SYSDTA kennzeichnen                                                |
| ESCAPE                 | Prozedurablauf unterbrechen                                                      |
| EXIT-PROCEDURE         | Prozedurablauf beenden und Rückkehr zur zuletzt verlassenen Prozedurdatei        |
| HOLD-JOB               | Auftrag (Job) anhalten                                                           |
| HOLD-PROCEDURE         | unterbricht Prozedurablauf und ermöglicht die Kommandoingabe vom Datensichtgerät |

| <b>Kommando</b>              | <b>Kurzbeschreibung</b>                                                  |
|------------------------------|--------------------------------------------------------------------------|
| HOLD-SPOOLOUT                | Spooloutauftrag anhalten                                                 |
| HOLD-SUBSYSTEM               | Wartezustand für Subsystem                                               |
| INFORM-OPERATOR              | Nachricht an die Konsole senden                                          |
| INTR                         | "INTR-Ereignis" an Programm senden                                       |
| LOAD-EXECUTABLE-PROGRAM      | Programm laden                                                           |
| LOAD-PROGRAM                 | Programm laden                                                           |
| LOGON                        | Auftrag (Job) einleiten                                                  |
| MODIFY-ACCOUNTING-PARAMETERS | legt Abrechnungssätze und Satzerweiterungen für die Accountingdatei fest |
| MODIFY-TASK-CATEGORIES       | begrenzt die Anzahl aktiver Tasks                                        |
| ON                           | bedingte Ausführung einer Kommandofolge                                  |
| PROCEDURE                    | Prozedurdateimerkmale festlegen                                          |
| REMOVE-CJC-ACTION            | Wirksamkeit einer CJC-Kommandofolge aufheben                             |
| RESTART                      | Programm bei Fixpunkt starten                                            |
| RESUME-JOB                   | Wartezustand für Anwenderauftrag aufheben                                |
| RESUME-PROCEDURE             | unterbrochenen Prozedurablauf fortsetzen                                 |
| RESUME-SUBSYSTEM             | Wartezustand für Sybssystem aufheben                                     |
| RTI                          | Rückkehr zur unterbrochenen Prozedur                                     |
| SEND-MSG                     | Nachricht an Konsole oder Programm (STXIT-Routine) senden                |
| SET-JOB-STEP                 | beendet SPIN-OFF                                                         |
| SET-LOGON-PARAMETERS         | Dialog- oder Batchauftrag einleiten                                      |
| SET-SPACE-SATURATION-LEVEL   | Festlegen der Speicher-Sättigungsstufen auf einen Pubset                 |
| SHOW-ACCOUNTING-STATUS       | informiert über das Abrechnungssystem                                    |
| SHOW-FILE                    | Datei auf Bildschirm ausgeben                                            |
| SHOW-SERSLOG                 | gibt Informationen über SERSLOG aus                                      |
| SHOW-SUBSYSTEM-STATUS        | Informationen über Zustand von Subsystemen anfordern                     |
| SKIP                         | bedingt (Auftragungsschalter) springen                                   |
| SKIP-COMMANDS                | Zu Sprungziel verzweigen (abhängig von Schaltern oder JV)                |
| SKIPJV                       | bedingt (Jobvariable) springen                                           |
| SKIPUS                       | bedingt (Benutzerschalter) springen                                      |
| SPMGT                        | Speicherplatz verwalten                                                  |
| START-ACCOUNTING             | Abrechnungssystem einschalten                                            |

| <b>Kommando</b>          | <b>Kurzbeschreibung</b>                            |
|--------------------------|----------------------------------------------------|
| START-EXECUTABLE-PROGRAM | Programm laden und starten                         |
| START-PROGRAM            | Programm laden und starten                         |
| START-SERSLOG            | aktiviert SERSLOG                                  |
| START-SUBSYSTEM          | Subsystem laden und initialisieren                 |
| STEP                     |                                                    |
| STOP-ACCOUNTING          | Abrechnungssystem ausschalten                      |
| STOP-SERSLOG             | SERSLOG beenden                                    |
| STOP-SUBSYSTEM           | Subsystem beenden                                  |
| WAIT                     | bedingte Wartezeit (Stapelauftrag) angeben         |
| WAIT-EVENT               | Auftrag ereignisabhängig in Wartezustand versetzen |
| WHEN                     | Stapelauftrag bedingt (Benutzerschalter) anhalten  |

## 13.2 Ereigniscodes

Folgende Tabelle ordnet den STXIT-Ereignisklassen die zugehörigen Unterbrechungsereignisse und die Ereigniscodes zu:

| STXIT-Ereignisklasse            | Unterbrechungsereignis                                                                                                                                                                                                                                                              | Ereigniscode                                                                           |
|---------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|
| Programmfehler                  | unzulässiger SVC<br>unzulässiger Operationscode<br>Datenfehler<br>Exponentenüberlauf<br>Divisionsfehler<br>Mantisse = 0<br>Exponentenunterlauf<br>Dezimalüberlauf<br>Festpunktüberlauf                                                                                              | X' 04'<br>X' 58'<br>X' 60'<br>X' 64'<br>X' 68'<br>X' 6C'<br>X' 70'<br>X' 74'<br>X' 78' |
| Intervallzeitgeber für CPU-Zeit | "SETIC-Intervall" abgelaufen für CPU-Zeit                                                                                                                                                                                                                                           | X' 20'                                                                                 |
| Intervallzeitgeber für Realzeit | "SETIC-Intervall" abgelaufen für Realzeit                                                                                                                                                                                                                                           | X' A0'                                                                                 |
| Ende Programmlaufzeit           | Ende der Programmlaufzeit                                                                                                                                                                                                                                                           | X' 80'                                                                                 |
| nicht behebbarer Programmfehler | privilegierter SVC<br>Zugriff auf eine nicht vorhandene Speicherseite<br>privilegierte Operation<br>Adressenfehler<br>XA-Fehler (falscher Adressierungsmodus)<br>Realtimer (Condition Error)<br>Ausrichtungsfehler<br>Validierungsfehler<br>nicht behebbarer Vektorprozessor Fehler | X' 08'<br>X' 48'<br>X' 54'<br>X' 5C'<br>X' 9C'<br>X' A4'<br>X' AC'<br>X' B0'<br>X' B4' |
| Mitteilung an das Programm      | Kommando                                                                                                                                                                                                                                                                            | X' 44'                                                                                 |
| ESCPBRK                         | BREAK/ESCAPE (über Tasten)                                                                                                                                                                                                                                                          | X' 84'                                                                                 |
| ABEND                           | Systemfehler, Leitungsverlust<br>START-EXECUTABLE-PROGRAM,<br>LOAD-EXECUTABLE-PROGRAM, ABEND,<br>LOGOFF, CANCEL-JOB<br>Adress-Übersetzungsfehler wegen Hardwarefehler<br>Hardwarefehler (CPU)<br>erzwungenes Entladen eines Subsystems<br>(Systemverwaltung)                        | X' 88'<br>X' 8C'<br>X' 94'<br>X' A8'<br>X' B8'                                         |
| Programmbeendigung              | TERM<br>CMD                                                                                                                                                                                                                                                                         | X' 90'<br>X' 98'                                                                       |
| SVC-Unterbrechung               | SVC-Aufruf eines angegebenen SVCs                                                                                                                                                                                                                                                   | X' 50'                                                                                 |
| Hardwarefehler                  | Ein-/Ausgabefehler bei Data-In-Virtual-Technik                                                                                                                                                                                                                                      | X' 28'                                                                                 |

---

# Fachwörter

## Ablaufüberwachung

%CONTROLn, %INSERT und %ON sind Kommandos zur Ablaufüberwachung. Kommt der Programmablauf an eine Anweisung oder einen Befehl 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 Befehle oder Anweisungen protokolliert werden sollen. Im Standardfall können Sie den Programmablauf am Bildschirm mitverfolgen.

## Adressoperand

ist ein Operand, mit dem Sie eine Speicherstelle oder einen Speicherbereich adressieren. Sie können virtuelle Adressen, Datennamen, Anweisungsnamen, Source-Referenzen, Schlüsselwörter, komplexe Speicherreferenzen, eine C-Qualifikation (maschinennahes Testen) oder eine PROG-Qualifikation (symbolisches Testen) angeben. Die Speicherstelle bzw. der Speicherbereich liegen entweder im geladenen Programm oder in einem Speicherabzug in einer Dump-Datei.

Wenn ein Name in Ihrem Programm mehrfach vergeben wurde und somit kein eindeutiger Bezug auf eine Adresse gewährleistet ist, können Sie ihn mit Bereichsqualifikationen oder über eine *kennzeichnung* (COBOL) eindeutig der gewünschten Adresse zuordnen.

## Adressierungsmodus

AID übernimmt den Adressierungsmodus des Testobjekts (24-Bit-Adressen oder 31-Bit-Adressen). Sie können mit AID auch Programme testen, die aus Moduln mit unterschiedlichem Adressierungsmodus gebunden wurden. Im Systeminformationsfeld %AMODE ist stets der aktuelle Adressierungsmodus eingetragen. Ändern können Sie den Adressierungsmodus mit %MOVE %MODE{24|31} INTO %AMODE, anschauen mit %DISPLAY %AMODE.

### **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 Speicherreferenzen ohne Angabe einer Basisqualifikation ansprechen können.

Er umfasst den nicht-privilegierten Teil des virtuellen Speichers in Ihrer Task, der vom Programm samt allen konnektierten Subsystemen belegt ist, oder den entsprechenden Bereich in einem Speicherabzug.

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

### **AID-Ausgabedateien**

sind die Dateien, in die Sie sich die Ausgaben der Kommandos %DISASSEMBLE, %DISPLAY, %HELP, %SDUMP und %TRACE schreiben lassen können. Die Dateien werden in den Ausgabekommandos über ihre Linknamen F0 bis F7 angesprochen (siehe %OUT und %OUTFILE).

In die Datei, die dem Linknamen F6 zugewiesen wurde, werden die REP-Sätze geschrieben (siehe %AID REP=YES und %MOVE).

Es gibt drei Wege, eine Ausgabedatei anzulegen bzw. eine Ausgabedatei zuzuweisen:

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

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

### **AID-Eingabedateien**

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

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

## AID-Literale

AID stellt Ihnen Zeichen-Literale und numerische Literale zur Verfügung (siehe Kapitel „AID-Literale“ auf Seite 113):

|                                            |                   |
|--------------------------------------------|-------------------|
| {C'x...x'   'x...x'C   'x...x'   U'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-Adressinterpretation

Indirekte Adressen, d.h. Adressen vor einem Pointer-Operator, werden im Standardfall entsprechend dem gerade gültigen Adressierungsmodus interpretiert. Mit %AINT können Sie von der Standard-Adressinterpretation abweichen und festlegen, ob AID bei indirekter Adressierung mit 24-Bit-Adressen oder mit 31-Bit-Adressen arbeiten soll.

## AID-Standard-Arbeitsbereich

Das ist der nicht-privilegierte Teil des virtuellen Speichers in Ihrer Task, der vom Programm samt allen konnektierten Subsystemen belegt ist. Ohne Vereinbarung mit %BASE und ohne Angabe einer Basisqualifikation gilt der AID-Standard-Arbeitsbereich.

## Aktuelle Aufrufhierarchie

ist der Stand der Unterprogrammverschachtelung an der Unterbrechungsstelle. Sie reicht von der Unterprogrammebene, auf der das Programm unterbrochen wurde über die durch CALL-Anweisungen verlassenen Unterprogramme mittlerer Hierarchiestufen bis zum Hauptprogramm. Sie wird mit %SDUMP %NEST ausgegeben.

## Aktuelle CSECT

ist die CSECT, in der das Programm unterbrochen wurde. Die STOP-Meldung gibt ihren Namen aus.

## Aktuelles Programm

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

## Aktueller Programmteil

ist der Programmteil, in dem das Programm unterbrochen wurde. Die STOP-Meldung gibt seinen Namen aus.

### Anweisungsname

bezeichnet einen im Quellprogramm vergebenen Namen, mit dem in AID eine ausführbare Anweisung angesprochen werden kann. Das sind Marken oder Namen von Haupt- oder Unterprogrammen. Hierzu ist in den LSD-Sätzen eine Adresskonstante hinterlegt, die die Adresse der ersten Anweisung nach der Marke bzw. im Haupt- oder Unterprogramm enthält. Genauer ist es die Adresse des ersten Befehls, der zu der ersten ausführbaren Anweisung nach einer Marke bzw. im Haupt- oder Unterprogramm erzeugt wurde.

### Attribute

Jedes Speicherobjekt hat bis zu sechs Attribute:

Adresse, Name (opt), Inhalt, Länge, Speichertyp, Ausgabetyt.

Mit Selektoren können Sie auf Adresse, Länge und Speichertyp zugreifen. Über den Namen findet AID in den LSD-Sätzen alle zugehörigen Attribute, um das zugehörige Speicherobjekt richtig zu interpretieren.

Adresskonstanten und Konstanten aus dem Quellprogramm haben nur bis zu fünf Attribute:

Name (opt), Wert, Länge, Speichertyp, Ausgabetyt.

Sie haben keine Adresse. Beim Ansprechen einer Konstanten greift AID nicht auf ein Speicherobjekt zu, sondern setzt nur den Wert der Konstanten ein.

### Ausgabetyt

Attribut eines Speicherobjekts, das bestimmt, wie der Speicherinhalt von AID ausgegeben wird. Jedem Speichertyp ist ein Ausgabetyt zugeordnet. In [Kapitel „Schlüsselwörter“ auf Seite 125](#) sind die AID-spezifischen Speichertypen samt zugehörigen Ausgabetyten aufgelistet. Für die Datentypen in der jeweiligen Programmiersprache gilt eine entsprechende Zuordnung. Eine Typmodifikation bei %DISPLAY und %SDUMP bewirkt eine Änderung des Ausgabetyts.

### Basisqualifikation

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

### Bereichsgrenzen

Jedem Speicherobjekt ist ein bestimmter Bereich zugeordnet, der bei Datennamen und Schlüsselwörtern durch die Attribute Adresse und Länge festgelegt ist. Bei virtuellen Adressen liegen die Bereichsgrenzen zwischen V'0' und der letzten Adresse des virtuellen Speichers (V'7FFFFFFF'). Bei Bereichsqualifikationen ergeben sich die Bereichsgrenzen aus Anfangs- und Endadresse des damit bezeichneten Programnteils (siehe [Kapitel „Adressierung in AID“ auf Seite 69](#)).



### Bereichsqualifikationen

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

### 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 einem Datennamen sprechen Sie Variablen, Konstanten, Strukturen, Tabellen und Struktur- oder Tabellenelementen beim symbolischen Testen an. Auf Elemente von Tabellen können Sie sich wie in der jeweiligen Programmiersprache über einen Index beziehen.

### Datentyp

Gemäß dem im Quellprogramm deklarierten Datentyp ordnet AID allen Datenelementen einen AID-Speichertyp zu:

- Binärstring (≙ %X)
- Character (≙ %C oder %UTF16)
- numerisch, wobei nicht alle Datentypen, die in der jeweiligen Programmiersprache numerisch behandelt werden, auch für AID vom Speichertyp numerisch sind (siehe hierzu die einzelnen sprachspezifischen AID-Handbücher [2]-[6]).

Der zugeordnete Speichertyp bestimmt, wie das Datenelement von %DISPLAY ausgegeben, von %MOVE oder %SET übertragen bzw. überschrieben und wie es in der Bedingung eines Subkommandos verglichen wird.

### **Eingabepuffer**

AID hat einen internen Eingabepuffer. Reicht er für die Aufnahme der Eingabe eines Kommandos nicht aus, wird das Kommando mit einer Fehlermeldung als zu lang abgewiesen. Die gewünschte Operation müssen Sie auf zwei Kommandos aufteilen, damit AID sie durchführen kann.

### **ESD/ESV**

External Symbol Dictionary bei OMs / External Symbols Vector bei LLMs ist das Verzeichnis der Externbezüge eines Moduls. Es wird vom Compiler erstellt. Hierin sind unter anderem Informationen über CSECTs, DSECTs und COMMONs enthalten. Der Binder greift auf dieses Verzeichnis zu, wenn er die Objekt-Strukturliste erzeugt.

### **globale Einstellungen**

AID stellt Ihnen Kommandos zur Verfügung, mit denen Sie das Verhalten von AID Ihren Testerfordernissen anpassen können, die Ihnen die Adressierung erleichtern oder Schreibaarbeit ersparen. Die Voreinstellungen gelten während der gesamten Testsitzung (siehe %AID, %AINT, %BASE und %QUALIFY).

### **Hauptprogramm**

wird in diesem Handbuch als Sammelbegriff für das Programm (COBOL), die Funktion (main bei C++/C) oder die externe Prozedur (PL/I) verwendet, die beim Anlaufen des Programms vom System gestartet wird.

### **Index**

ist ein Teil eines Adressoperanden. Mit einem Index wird die Position eines Vektorelements bestimmt. Er kann wie in der Programmiersprache angegeben werden oder durch einen arithmetischen Ausdruck, aus dem AID den Wert des Index errechnet.

### **Kommandofolge**

Mehrere Kommandos werden mit Semikolon (;) zu einer Folge verbunden, die von links nach rechts abgearbeitet wird. Wie im Subkommando darf eine Kommandofolge AID- und BS2000-Kommandos enthalten. In Kommandofolgen nicht zugelassen sind die AID-Kommandos %AID, %BASE, %DUMPFILe, %HELP, %OUT, %QUALIFY und die im Anhang aufgelisteten BS2000-Kommandos.

Enthält eine Kommandofolge eines der Kommandos zur Ablaufsteuerung, wird die Kommandofolge an der Stelle abgebrochen und das Programm gestartet (%CONTINUE, %RESUME, %TRACE) oder angehalten (%STOP). Nachfolgende Kommandos aus der Kommandofolge werden nicht mehr ausgeführt.

### Kommandomodus

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

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

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

### Konstante

Eine Konstante repräsentiert einen Wert, der nicht über eine Adresse im Programmspeicher hinterlegt ist.

Zu den Konstanten gehören die im Quellprogramm definierten Konstanten, die Ergebnisse von Längenselektion, Längenfunktion und Adressselektion sowie die Anweisungsnamen und die Source-Referenzen.

Eine Adresskonstante repräsentiert eine Adresse. Adresskonstanten sind Anweisungsnamen, Source-Referenzen und das Ergebnis einer Adressselektion.

Eine Adresskonstante kann in einer komplexen Speicherreferenz nur vor einem Pointer-Operator (->) eingesetzt werden.

### 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 [Abschnitt „Ketten“ auf Seite 62](#)).

### 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, das ist die Aufrufhierarchie zur aktuellen Programmunterbrechungsstelle.

### **LSD**

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

### **Namensraum**

umfasst alle zu einem Programmteil in den LSD-Sätzen verzeichneten Daten-  
namen.

### **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**

Begriff, der in Fortran für das gebraucht wird, was in anderen Programmiersprachen mit Übersetzungseinheit bezeichnet wird. Die Programmeinheit können Sie mit der S-Qualifikation ansprechen.

### **Programmteil**

ist ein Sammelbegriff für den Teil eines Programms, der mit einer Bereichsqualifikation angesprochen werden kann. Jede Programmiersprache hat dafür eigene Bezeichnungen, die in den sprachspezifischen AID-Handbüchern beschrieben sind.

### **Programmzustand**

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

1. Das Programm steht.

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

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 Speicherreferenz ansprechen, die nicht im AID-Arbeitsbereich liegt oder außerhalb des aktuellen Haupt- oder Unterprogramms oder darin nicht eindeutig ist. Die Basisqualifikation gibt an, ob die Speicherreferenz im geladenen Programm oder in einem Speicherabzug liegt. Eine Bereichsqualifikation gibt an, in welchem Programmteil eine Speicherreferenz 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 zu einer virtuellen Adresse eine Bereichsqualifikation angegeben wird.

### Source-Referenz

bezeichnet eine ausführbare Anweisung. Sie wird mit S'nummer/name' angegeben. *nummer/name* wird vom Compiler erzeugt und in den LSD-Sätzen hinterlegt.

### Speicherobjekt

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

### Speicherreferenz

Mit einer Speicherreferenz sprechen Sie ein Speicherobjekt an. Es gibt einfache und komplexe Speicherreferenzen. Einfache Speicherreferenzen sind virtuelle Adressen, Namen, zu denen AID sich die Adresse aus den LSD-Informationen holen kann, und Schlüsselwörter. Anweisungsnamen und Source-Referenzen sind in den AID-Kommandos %CONTROLn, %DISASSEMBLE, %INSERT, %JUMP, %REMOVE und %TRACE als Speicherreferenz erlaubt, obwohl sie nur Adresskonstanten sind.

Mit den komplexen Speicherreferenzen geben Sie AID eine Vorschrift an, wie die gewünschte Adresse errechnet werden soll und welcher Typ und welche Länge gelten sollen. Folgende Operationen können in einer komplexen Speicherreferenz vorkommen: Adressversatz, indirekte Adressierung, Typmodifikation, Längenmodifikation und Adressselektion.

### Speichertyp

ist entweder der Datentyp, der im Quellprogramm festgelegt wurde oder der durch Typmodifikation gewählte. AID kennt die Speichertypen %X, %C, %E, %P, %D, %F, %A (siehe %SET und [Kapitel „Adressierung in AID“ auf Seite 69](#) und [Kapitel „Schlüsselwörter“ auf Seite 125](#)).

### Subkommando

ist ein Operand der Überwachungskommandos %CONTROLn, %INSERT oder %ON. Ein Subkommando besteht aus einem Kommandoteil, dem wahlweise ein Name und eine Bedingung vorangestellt sein kann. 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 in [Kapitel „Subkommando“ auf Seite 51](#) beschrieben.

Der Kommandoteil des Subkommandos wird dann ausgeführt, wenn die Überwachungsbedingung des entsprechenden Kommandos (*kriterium*, *testpunkt*, *ereignis*) zutrifft und die eventuell definierte Ausführungsbedingung erfüllt ist.

### Übersetzungseinheit

Der Teil eines Programms, der als eine Einheit übersetzt wurde. Bei Fortran wird hierfür der Begriff Programmeinheit verwendet. Die Übersetzungseinheit können Sie mit der S-Qualifikation ansprechen.

### Unterbrechungsstelle

Die Adresse, an der ein Programm unterbrochen wurde, wird Unterbrechungsstelle genannt. Aus der STOP-Meldung können Sie entnehmen, an welcher Adresse und in welchem Programmteil die Unterbrechungsstelle liegt. Dort wird das Programm fortgesetzt. Für COBOL- und FOR1-Programme können Sie mit %JUMP eine andere Fortsetzungsadresse vereinbaren.

### Unterprogramm

wird in diesem Handbuch als Sammelbegriff für Funktionen (C++/C, Fortran, COBOL), Prozeduren (PL/I) oder Programme (COBOL) verwendet, die in der Aufrufhierarchie dem Hauptprogramm untergeordnet sind.

### Zeichenkonvertierungsfunktionen

AID stellt zwei Funktionen zur Zeichenkonvertierung %C() und %UTF16() zur Verfügung.

Die Funktion %UTF16() wandelt Strings von einer 1-Byte-EBCDIC-Codierung in die UTF16-Codierung um, die Funktion %C realisiert die Konvertierung umgekehrt.

---

# Literatur

Die Handbücher finden Sie im Internet unter <http://manuals.ts.fujitsu.com>. Handbücher, die mit einer Bestellnummer angezeigt werden, können Sie auch in gedruckter Form bestellen.

- [1] **AID (BS2000)**  
**Testen auf Maschinencode-Ebene**  
Benutzerhandbuch
  
- [2] **AID (BS2000)**  
Advanced Interactive Debugger  
**Testen von COBOL-Programmen**  
Benutzerhandbuch
  
- [3] **AID (BS2000)**  
Advanced Interactive Debugger  
**Testen von FORTRAN-Programmen**  
Benutzerhandbuch
  
- [4] **AID (BS2000)**  
Advanced Interactive Debugger  
**Testen unter Posix**  
Benutzerhandbuch
  
- [5] **AID (BS2000)**  
Advanced Interactive Debugger  
**Testen von ASSEMBH-Programmen**  
Benutzerhandbuch
  
- [6] **AID (BS2000)**  
Advanced Interactive Debugger  
**Testen von C/C++-Programmen**  
Benutzerhandbuch
  
- [7] **AID (BS2000)**  
Advanced Interactive Debugger  
**Tabellenheft**  
Benutzerhandbuch

- [8] **BS2000 OSD/BC  
Kommandos**  
Benutzerhandbuch
- [9] **BS2000 OSD/BC  
Einführung in die Systembetreuung**  
Benutzerhandbuch
- [10] **BS2000 OSD/BC  
Makroaufrufe an den Ablaufteil**  
Benutzerhandbuch
- [11] **LMS (BS2000)**  
SDF-Format  
Benutzerhandbuch
- [12] **BINDER**  
**Binder in BS2000**  
Benutzerhandbuch
- [13] **BLSSERV**  
**Bindelader-Starter in BS2000**  
Benutzerhandbuch
- [14] **SDF (BS2000)**  
**Dialogschnittstelle SDF**
- [15] **XHCS**  
**8-bit-Code- und Unicode-Unterstützung im BS2000**  
Benutzerhandbuch



---

# Stichwörter

\*OMF-Datei 39  
&Trace 144  
%• 67, 101, 133  
%\*subkdoname 53, 54, 67, 87, 133  
%A 126  
%AID 23, 31, 113  
%AINT 17, 31  
%AMODE 17, 131  
%AR 18, 129  
%ASC 18, 131  
%AUD1 131  
%BASE 22, 31, 52, 70  
%C 125  
%CC 131  
%CCSN 24, 114  
%CLASS5 / %CLASS6 87, 89, 130  
%CONTINUE 28, 48  
%CONTROL 144  
%CONTROLn 21, 27, 51, 53, 62, 69, 73, 86, 130, 134, 141  
%D 126  
%DISASSEMBLE 29, 31, 69, 85, 107, 130  
%DISPLAY 17, 29, 31, 54, 69, 72, 86, 102, 107, 125, 129, 130, 133  
%DS 18, 131  
%DUMPFIL 30, 31, 70  
%F 126  
%FALSE 133  
%FIND 30, 69, 72, 130, 142  
%FR 129  
%H 126  
%HELP 31, 32, 107  
%HLLOC 21, 131  
%INSERT 27, 51, 62, 65, 69, 85, 141  
%JUMP 28, 85  
%LINK 49, 131  
%LOC 21, 36, 131  
%LPOV 49, 63, 66, 137  
%MAP 36, 131  
%MODE24/31 134  
%MOVE 17, 29, 54, 69, 72, 79, 86, 104, 129, 142  
%MR 129  
%n 129  
%nAR 18, 71, 87, 129  
%nD 129  
%nE 129  
%NEST 29, 134  
%nG 71, 130  
%nGD 130  
%NL 134  
%NP 133  
%nQ 129  
%ON 27, 51, 62, 63, 65, 135, 139  
%ON %LPOV 131  
%ON %SVC 53  
%ON %WRITE(...) 32, 72, 135, 141  
%OUT 31, 107  
%OUTFILE 31  
%P 126  
%PC 28, 129  
%PCB 129, 131  
%PCBLST 131  
%PM 131  
%QUALIFY 32, 52, 70  
%REMOVE 27, 67, 85, 140  
%RESUME 28, 48  
%S 60, 92, 128, 142  
%SDUMP 21, 29, 31, 69, 73, 107, 134  
%SET 29, 54, 57, 69, 79, 86, 125, 129, 133, 142

%SORTEDMAP 36, 131

%STOP 28, 48

%SVC 53, 66, 137

%SX 60, 92, 128, 142

%SYMLIB 20, 21, 30, 32, 43

%TITLE 31

%TRACE 21, 28, 31, 48, 53, 69, 73, 86, 107,  
130, 134, 141, 142

%TRUE 133

%UTF16 24, 58, 78, 104, 114, 125

%WRITE 32, 62, 72, 135, 141

%X 125

%Y 126

### A

Abarbeitung Bedingungsoperatoren 56

Ablaufsteuerung 22, 25, 28

Ablaufüberwachung 21, 22, 25, 27, 32, 69, 142

Ablaufverfolgung 21, 28

Adressierungsmodus 17, 18, 31, 91, 131, 134

Adressinterpretation bei indirekter  
Adressierung 134

Adresskonstante 69, 85, 91, 92, 93, 101, 102,  
104

Adressoperand 21, 32, 52, 69, 70, 71, 72

Adresselektion 21, 22, 79, 84, 88, 89, 90, 92,  
101, 102

Adressversatz 14, 71, 79, 80, 87, 89, 90, 100,  
104

AID-Arbeitsbereich 21, 22, 30, 31, 44, 108, 109

AID-Gleitpunktregister 130

AID-interner Eingabepuffer 58, 65

AID-Kommandos 47  
Übersicht 25

AID-Literal 22, 29, 57, 95, 113, 126  
alphanumerisch 113  
numerisch 60, 121

AID-Register 87, 130

AID-Standard-Arbeitsbereich 21

AIDSYS 15

aktuelle Aufrufhierarchie 29, 134

aktueller Programmteil 71

ALET-Qualifikation 18, 71, 80, 131

anderes Ausgabeformat 95

Anpassung Speichertypen

%SET 95, 96

Vergleich 57

Anweisungsname 21, 22, 37, 73, 82, 85, 86, 88,  
92, 101

Apostroph 113

AR-Modus 18, 131

arithmetischer Ausdruck 88, 100, 129

ASC-Modus 18, 131

ASCII 57

Assembler 71, 73, 85, 140

Assembler-Notation 29

Attribute 79, 82

C-/COM-Qualifikation 80

Datennamen 82

Schlüsselwörter 87

Speicherklassen 130

virtuelle Adresse 81

Aufbereitung für Systeminformationen 130

ausdruck 88, 89, 99

ausführbarer Teil 20

Ausgabe

auf SYSLSST 108

von Speicherinhalten und Literalen 29

Ausgabedatei 17, 31, 108

Ausgabekommando 26, 31

Ausgabemedium 31, 107

Ausgabetypp 79, 125

Speichertyp-Zuordnung 125

verändern 96

virtuelle Adresse 125

Ausgeben Durchlaufzähler 54

automatisierte Testabläufe 51

### B

BASED-Variable 92

Basisqualifikation 21, 30, 31, 32, 52, 70, 73, 80,  
87, 96

Basisregister und Distanz 128

Bedingung in einem Subkommando 55

Befehlscode 22, 85

Befehlstypen 134

Befehlszähler 28, 87, 129

- Bereichsgrenzen 71, 79, 84, 89, 96, 98, 103, 104  
     CSECT/COMMON 80  
     Dateiname 83  
     Schlüsselwort 87  
     virtuelle Adresse 81  
 Bereichsqualifikation 71  
 Bibliothek 19, 20, 30, 32, 35, 37, 40, 43, 44  
 Binär-Literal 120  
 Binärvergleich 57  
 BIND-Makro 41, 42, 140  
 Bindelademodul 39, 40, 41  
     LSD-Sätze 39  
 Bindemodul 20, 40, 41, 44  
     LSD-Sätze 39  
 Binden 36, 40  
 BINDER 20, 35, 36, 37, 39, 41, 43  
 Bindestrich 23, 31, 45, 46, 53  
 Bit-Literal 120  
 BLK-Qualifikation 73  
 Boolesche Operatoren 56  
 BS2000-Kommandos in Kommandofolge 48  
 Byte-Offset 89  
 Byte-Reihenfolge  
     invertieren 126
- C**
- C-Qualifikation 72, 80, 88, 89, 95, 98, 102  
 C++/C 71, 85, 88, 92, 93, 104, 139  
 Character-Literal 57, 113  
     numerischer Inhalt 114  
 Character-Vergleich 57  
 CMD-Makro 16, 47  
 COBOL 28, 57, 71, 72, 73, 82, 83, 85, 86, 139  
 COBOL85 28  
 COM-Qualifikation 72, 80, 88, 98, 102  
 COMMON 20, 21, 23, 30, 36, 72, 80, 109, 131  
 Common Memory Pool, 143  
 Compiler-Option 39  
 Condition Code 131  
 Contingencies 144  
 CSECT 20, 21, 23, 30, 36, 72, 80, 109, 131, 140  
     maskiert 43  
     Umbenennen 35
- CSECTs  
     gleichnamig im LLM 36, 41, 43
- D**
- Datei-Ausgabe 108  
 Dateiname 21, 73, 82, 88, 89, 95, 98, 102  
     Konstante 82  
 Datenraum 18, 71, 80  
 Datenschutz 19  
 DBL 41, 72  
 Dezimalpunktzahl 122  
 Dump-Datei 17, 20, 22, 30, 31, 32, 44, 70, 74,  
     109  
 Durchlaufzähler 14, 53, 87, 133
- E**
- EBCDI-Code 57  
 Edit-Lauf 41  
 einfache Speicherreferenzen 78, 85  
 Eingabe  
     von Operanden 45  
     von Qualifikationen 70  
 Eingabedatei 17  
 Entry 23, 104  
 Environment 70  
 ereignis 25, 27, 32, 51, 62, 65, 66, 67, 135, 139  
 ereignis-Tabelle 137  
 Ereigniscode 137, 148  
 Erzeugung der LSD-Sätze 39  
 ESD 20, 37  
 ESV 20, 37, 40, 42  
 exponent 60, 123
- F**
- Fehler in Subkommandos 52  
 Fehlverhalten 144  
 FIFO-Prinzip 135  
 Folgezeile  
     Dialog 46  
     Prozedurdateien 46  
 FOR1 28  
 Fortran 71, 73, 82, 85, 133, 139  
 Fortsetzen 28  
 Fortsetzungsadresse 28, 69

funktion 104

### G

Ganzzahl 59, 60, 83, 98, 100, 121, 126  
gleichnamige CSECTs im LLM 36, 41, 43  
Gleitpunktregister 87, 129  
Gleitpunktzahl 59, 123  
Groß-/Kleinschreibung 23, 31

### H

Hardcopy-Ausgabe 108  
Hardware-Audit-Tabelle 131  
high-level 21

### I

ILCS 139  
INCLUDE-MODULES 43  
INCLUSION-DEFAULT 41  
Index 99, 129  
    für %PC und %PCB 129  
    und Subskript 83  
Index-  
    Basisregister und Distanz 128  
indirekte Adressierung 31, 80, 81, 86, 88, 91,  
    134  
    auf symbolischer Ebene 91, 92  
Inhalt einer Speicherreferenz 100  
Inhaltsoperator 88, 91, 93  
Interpretation  
    als Adresse 95  
    als Ganzzahl 95  
    von indirekten Adressangaben 31  
ISP 17

### K

Ketten von Subkommandos 62, 65  
Kleinbuchstaben 23, 31, 113  
Kommandofolge 47, 51  
Kommandoformat 45  
Kommandointerpreter 16  
Kommentare 46  
komplexe Speicherreferenzen 21, 22, 71, 78, 88,  
    89, 98, 104, 125, 130

Konstanten 22, 78, 79, 82  
    im Quellprogramm definiert 82  
Kontext 21, 32, 36, 72, 131  
Kontextqualifikation 72  
Kopfzeile 109  
kriterium 27, 134, 135, 141  
Kurzform %• 53, 101, 133

### L

Ladeeinheit 21, 40, 41, 42, 44, 131  
Laden 36  
    mit LSD-Sätzen 40  
    ohne LSD-Sätze 40  
Länge  
    AID-Kommando 46  
    Kommandofolge 47  
Längenfunktion 22, 89, 99, 100  
Längenmodifikation 71, 79, 83, 87, 88, 96, 98,  
    125  
    Wert 98  
Längenselektion 22, 89, 98, 100, 102  
    Vektor 99  
LAST-SAVE 41  
Leerzeichen 45, 46, 57, 100, 114, 123, 125  
LIFO-Prinzip 52, 65  
Linkname 17, 30, 31, 44, 70, 73, 108  
LLM 30, 35, 36, 39, 41, 43  
    LSD-Sätze 39  
LOCAL#DEFAULT 72  
logische Variablen 57, 133  
logische Werte 133  
Lokalisierungsinformationen 21, 36, 131  
Löschen  
    gekettete Subkommandos 63, 67  
    geschachtelte Subkommandos 67  
    Subkommando 67  
low-level 21  
LSD-Sätze 20, 21, 22, 30, 35, 37, 39, 40, 41, 48,  
    82, 85, 140  
    nachladen 41, 43, 44  
    Subkommando 52

**M**

mantisse [60](#), [123](#)  
 maschinennah [21](#)  
 maschinennahe  
     Lokalisierungsinformationen [131](#)  
 maschinennahe Speicherreferenzen [72](#), [78](#), [80](#),  
     [88](#)  
 maschinennahes Testen [35](#), [39](#), [71](#), [134](#)  
 maskierte CSECTS [43](#)  
 medium-u-menge [107](#)  
     XFLAT [109](#)  
     XMAX [109](#)  
 Mehrzweckregister [87](#), [129](#)  
 Metasyntax [13](#)  
 Modifikation von Speicherinhalten [29](#)  
 Modifikations-Kommando [26](#)  
 MODIFY-LLM-ATTRIBUTES [40](#)  
 MODIFY-MODULE-ATTRIBUTES [35](#), [43](#)  
 MODIFY-SYMBOL-VISIBILITY [43](#)

**N**

nachgeladenes Segment [140](#)  
 Nachladen von LSD-Sätzen [43](#), [44](#)  
 Namen  
     aus dem Quellprogramm [37](#)  
     für Kommandos [45](#)  
     zugelassene Zeichen [23](#)  
 NATIONAL [58](#)  
 numerischer Vergleich [57](#)

**O**

Objekt-Strukturliste [20](#), [36](#)  
 OM [30](#), [40](#), [41](#), [43](#)  
     LSD-Sätze [39](#)  
 ONUNIT-Qualifikation [73](#)  
 openUTM [140](#)  
 Overlay [31](#), [32](#), [140](#)

**P**

PL/I [71](#), [85](#), [92](#), [104](#)  
 PLAM-Bibliothek [20](#), [30](#), [35](#), [37](#), [39](#), [40](#), [43](#), [44](#)  
 Pointer-Operator [21](#), [81](#), [84](#), [86](#), [87](#), [88](#), [90](#), [91](#),  
     [92](#), [102](#), [104](#), [128](#)  
     Mehrzweckregister [87](#), [129](#)

Privilegierung [19](#)  
 PROC-Qualifikation [73](#)  
 Process Control Block [131](#)  
 PROG-Qualifikation [73](#)  
 Program Counter [129](#)  
 Program Mask [131](#)  
 Programmfehler [135](#)  
 Programmraum [18](#)  
 Programmregister [57](#), [129](#)  
 Programmzustand [28](#)  
 Prolog [104](#)  
 Protokollieren  
     Befehle [28](#)  
 Prozedurdatei [46](#), [50](#)  
 Prozess-Ebene [129](#)  
 Prüfung  
     AID-Kommandos [47](#)  
     Bedingung [56](#)  
     Bereichsgrenzen [89](#)  
     LSD-Sätze [52](#)  
     Qualifikationen [52](#)  
     Speicherinhalt - Speichertyp [96](#)  
 Punkt [70](#), [89](#)

**Q**

Qualifikation [18](#), [22](#), [32](#), [36](#), [52](#), [70](#), [71](#), [73](#), [80](#)

**R**

Readme-Datei [11](#)  
 Redefinition [95](#)  
 REP [17](#), [26](#), [29](#), [31](#), [36](#), [108](#)  
 REPLACE-MODULES [43](#)  
 RESOLVE-BY-AUTOLINK [43](#)  
 Rückübersetzen [29](#)  
 RUN-TIME-VISIBILITY [43](#)

**S**

S-Qualifikation [73](#)  
 SAVE-LLM [41](#)  
 Schachteln von Subkommandos [65](#)  
 Schlüsselwörter [57](#), [78](#), [88](#), [89](#), [95](#), [98](#), [101](#), [102](#),  
     [125](#)  
     für Adressoperanden [87](#)  
     für AR-Modus [18](#)

### Schlüsselwörter (Forts.)

- für Durchlaufzähler 53
  - für ereignis 135
  - für kriterium 21, 134
  - für Lokalisierungsinformationen 21
  - für Register 129
  - für Speicherbereiche 130
  - für Speichertypen 125
  - für Taskinformationen 130
  - zur Adressinterpretation 17, 134
- Schreibüberwachung 27, 135
- SDF 17
- SDF-P-Kontrollflusskommandos 48
- Sedezimal-Literal 57, 119
- Sedezimalzahl 80, 98, 121
- Selektoren 79, 102
- Semantik prüfen 47
- signal() 140
- SKIP-COMMANDS 50
- Source-Referenz 20, 21, 22, 37, 72, 73, 82, 85, 86, 88, 92, 101
- Speicherabzug 15
- Speicherbedarf eines Programms 20
- Speicherklassen 87, 130
- Speicherobjekt 22, 70
- Speicherreferenzen 22, 56, 78, 89
  - einfache 22, 85
  - komplexe 22, 71, 88, 89, 98, 104, 125, 130
  - maschinennahe 72, 80
  - symbolische 72, 82
- Speicherstelle als Adresse 92
- Speichertyp 79, 95, 125
  - Ausgabety-Zuordnung 82, 96, 125
  - verändern 57, 95, 125
  - virtuelle Adresse 125
  - zur Adressinterpretation 88
  - zur Interpretation von
    - Maschinenbefehlen 128
- Speichertypen
  - Invertierung der Byte-Reihenfolge 126
- SPID-Qualifikation 18, 71, 80, 131
- Sprungziele in Prozeduren 45
- Stand des Durchlaufzählers 54
- Standard-Linkage 139
- Standard-Speichertyp %X 98
- Standardwert
  - für Subkommando 51
  - medium-u-menge 107
- START-LLM-CREATION 40
- START-LLM-UPDATE 40
- STOP-Meldung 28, 53
- Strukturkomponente 92
- Subkommando 27, 48, 51, 133
  - Bedingung 64, 129
  - ketten 62
  - Name 14, 53
  - schachteln 65
- SVC 27, 66, 135, 137, 142, 148
  - protokollieren 53
- symbolisch 21
- symbolische Adresse 21, 47, 66
- symbolische Ebene verlassen 83, 89
- symbolische Lokalisierungsinformationen 21, 131
- symbolische Speicherreferenzen 72, 78, 82
- symbolisches Testen 37, 41, 71, 134
- SYMCHARS 26
- Syntax prüfen
  - AID-Kommandos 47
- SYSCMD 15
- SYSLST 29, 108, 133
- SYSOUT 15, 29
- Systeminformationen 130

## T

- Taskzeile 109
- Terminal-Ausgabe 108
- Test-Privilegierung 19, 130
- TEST-SUPPORT 39
- Testebenen 21
- Testobjekt 20
- testpunkt 25, 27, 32, 51, 60, 62, 63, 65, 66, 67, 86, 97, 140, 141, 142
  - in Overlay-Segment 140
- Testpunkte in Common Memory Pools 143
- Typ Character 57
- Typmodifikation 57, 83, 88, 92, 95, 98, 102
- Typselektor 96, 102

Typverträglichkeit 29  
Bedingung 57

## U

überflüssige Qualifikationen 70  
Überlagerungsstruktur 31, 140  
Übersetzen 36  
Übertragen  
    %MOVE 29  
    %SET 29  
Überwachungsbedingung 27, 51  
Überwachungskommandos 25, 27, 32, 69, 142  
Umbenennen von CSECTs 35  
UNBIND-Makro 140  
UNCHANGED 41  
Unterbrechungsstelle 28

## V

Verändern Durchlaufzähler 54  
Vergleich 57  
Vergleichsart 57  
Vergleichsoperatoren 56  
Verwaltungsfunktionen 30  
Verwaltungskommando 26  
Verzeichnis der Externbezüge 37  
virtuelle Adresse 18, 32, 70, 71, 80, 83, 88, 89,  
    95, 98, 109, 110, 125  
virtueller Speicher 21, 70  
Vorqualifikation 32, 71  
Vorschubsteuerung 133

## W

Wechsel auf Maschinencode-Ebene 83, 89  
Wildcard 113, 119  
write-ereignis 25, 27, 32, 62, 65, 135, 141

## X

XFLAT 109  
XMAX 109  
XS-Programmierung 17, 134

## Z

Zähler 54, 133

Zeichendarstellung  
    Zeichensatz 24  
Zeichenfolge suchen 30  
Zielzeile 109  
Zugriff  
    auf Befehlscode 85  
    auf Daten 82  
Zugriffsregister 18, 87, 129  
Zusatzinformation 108

