1 Preface

AID (Advanced Interactive Debugger) is a powerful interactive debugging tool which runs under the operating system BS2000. Error diagnostics, debugging and short-term error recovery of all programs generated in BS2000 are considerably more rapid and more straightforward than other approaches, such as inserting debugging aid statements into a program, for example. AID is permanently available and can be easily adapted to the relevant programming language. Programs that are debugged using AID need not be recompiled; they can be used without LSD information and even with AID corrections in a production run immediately. The range of functions of AID and its debugging language (using AID commands) are primarily tailored to interactive applications. AID can, however, also be used in batch mode. AID provides the user with a wide range of options for monitoring and controlling execution, effecting output and modification of memory contents. It also lets you call up information on program execution and on using AID.

AID permits debugging both on the symbolic level of the relevant programming language and on machine code level. Symbolic debugging allows you to use the names defined in the source code to address statements, functions and data items and to use the source reference generated by the compiler to address statements which have no name. The BS2000 commands in the AID documentation are described in the EXPERT form of the SDF (System Dialog Facility) format. SDF is the dialog interface to BS2000. The SDF command language supersedes the previous (ISP) command language.

AID V2.1A runs under BS2000 V10 and above; for ESA support and write monitoring you need BS2000/OSD V1.0 or higher.

1.1 Target group

AID is intended for all software developers working in BS2000 with the programming languages ASSEMBLER, COBOL, Fortran, C, C++, or PL/I, especially those who wish to debug as well as correct programs This manual is primarily intended for those involved in debugging with AID on machine code level.

1.2 Structure of the AID documentation

The AID documentation consists of the AID Core Manual, the language-specific manuals for symbolic debugging, and the manual for debugging on machine code level. For experienced AID users there is also a Ready Reference [11], giving the syntax of all the commands and the operands with brief explanatory notes. It also includes the %SET tables and a comparison of AID and IDA. All the information the user requires for debugging can be found by referring to the manual for the particular language required and the core manual. The manual for debugging on machine code level can either be used as a substitute for or as a supplement to any of the language-specific manuals.

AID Core Manual [1]

The core manual provides an overview of AID and deals with facts and operands which are the same in all programming languages. The AID overview describes the BS2000 environment, explains basic concepts and presents the AID command set. The other chapters discuss preparations for testing; command input; the operands *subcmd*, *compl-memref* and *medium-a-quantity*; AID literals and keywords. The manual also contains messages, BS2000 commands invalid in command sequences, operands supported in this version only and a comparison of AID and IDA.

AID User Guides

The User Guides deal with the commands in alphabetical order and describe all simple memory references. Apart from the present manual,
AID - Debugging on Machine Code Level,
the available User Guides are:
AID - Debugging of COBOL Programs [2]
AID - Debugging of FORTRAN Programs [3]
AID - Debugging of PL/I Programs [4]
AID - Debugging of ASSEMBH Programs [5]
AID - Debugging of C/CC++ Programs [12]

In the language-specific manuals and in the manual for debugging on machine code level the commands are listed in alphabetical order. All memory references which are not complex are described in these manuals.

In the language-specific manuals, the description of the operands is tailored to the particular programming language involved. The user is expected to be familiar with the relevant language tools and compiler operation.

The manual for debugging on machine code level can be used for programs for which no LSD records exist or for which the information from symbolic debugging does not suffice for error diagnosis. When debugging on machine code level, use of the AID commands is independent of the programming language in which the program was written.

1.3 Changes made since AID V2.0A

The following extensions have been incorporated to provide ESA support:

– The ALET/SPID qualification can be used to access virtual addresses in a data area.

– Access registers can be read and modified via the keywords %0AR to %15AR.

– The keyword %DS[(ALET/SPID-qua)] returns information on data areas and on the associated SPIDs and ALETs.

– The keyword %ASC returns information on the ASC mode, which determines whether the AR mode is set and whether or not data areas are addressed.

– In addition to its usual content, the %TRACE log now includes the contents of access registers, the new ESA machine instructions, and an asterisk (*) before addresses from data areas.

– Addresses from data areas are identified by an asterisk (*) in the output of %DISPLAY.

Two new qualifications, CTX and COM, are included in chapter 4, “Machine-code-specific addressing“, and in the description of address operands. Additional information can be found in section 7.1 of the AID Core Manual.

In chapter 5, “AID Commands“, repetitive text that occurs in every address operand, e.g. information pertaining to qualifications, memory references, etc. has been reduced to a minimum. Although this means that you may initially need to refer to chapter 4, “Machine-code-specific addressing“, relatively often, you will find the manual more pleasant to work with once you are used to the product.

A description of the %SHOW command has been added. This command can be used to request information on current commands, test points, and events as well as the associated subcommands and operand values.

The keywords %AMODE, %LINK and %MAP have been added.
%DISPLAY %LINK shows the name of the segment that was dynamically loaded at the last successful %LPOV event (see %ON).
%DISPLAY %MAP can be used to obtain an overview of contexts, together with the COMMONs and CSECTs. In contrast to the keyword %SORTEDMAP, the output contains long names in unabbreviated form.

The keywords %IFR, %IMR, and %ISR have been **dropped**.

The context name and existing COMMON areas are now included in AID outputs.

%ON can be used to monitor write access to a memory location. The event %WRITE (memref) was introduced with AID V2.0B for this purpose. In contrast to the first version, an area of 64KB can now be monitored.
The order of processing has changed for the %ON command in cases where multiple events occur at the same time.

LMS correction statements are now generated in SDF format only. They can only be processed with an LMS version ≥ V 2.0A.

The format for the message key for AID messages has changed from *In* to *AID0n*. In addition, AID message texts are now also available in German. Messages can now be requested with `/HELP-MESSAGE-INFORMATION`. The AID command %HELP may be used in this version to reference messages with the old message key *In*.

1.4 Notational conventions

italics In the body of the text, operands are shown in *lowercase italics*.

boldprint Text to be highlighted is shown in **boldprint**.

This symbol marks points to be specially noted.

2 Metasyntax

The metasyntax shown below is the notational convention used to represent commands. The symbols used and their meanings are as follows:

UPPERCASE LETTERS
: String which the user must enter unchanged to select a particular function.

lowercase letters
: String identifying a variable for which the user has to insert any of the permissible operand values.

```
⎧ alternative ⎫
⎨     ...     ⎬
⎩ alternative ⎭
```

```
{ alternative | ... | alternative }
```

: Alternatives; one of these alternatives must be picked. The two formats have the same meaning.

[optional]
: Specifications enclosed in square brackets may be omitted.
In the case of AID command names, only the entire part in square brackets can be omitted; any other abbreviations cause a syntax error.

[...]
: Reproducibility of an optional syntactical unit. If a delimiter, e.g. a comma, must be inserted before any repeated unit, it is shown before the periods.

{...}
: Reproducibility of a syntactical unit which must be specified at least once. If a delimiter, e.g. a comma, must be inserted before any repeated unit, it is shown before the periods.

<u>Underscore</u>
: Underscoring designates the default value, which AID inserts if the user does not specify a value for an operand.

- A bullet (period in bold print) delimits qualifications, stands for a *prequalification* (see the %QUALIFY command), is the operator for a byte offset or is part of the execution counter or subcommand name. A bullet is entered from the keyboard using the key for a normal period. It is actually a normal period, but here it is shown in bold to make it stand out better.

3 Prerequisites for debugging

You will need the assembly listing or comparable documentation such as the locator map and/or object listing of the compiler as reference information. If no such information on structures is available, the linkage editor listing can be used as a substitute.

3.1 Compiling, linking and loading

You can debug on machine code level and make full use of the complete functionality of AID without having to specify any special options or operands when compiling, linking and loading a program: the compiler generates an ESD (External Symbol Dictionary) or ESV (External Symbol Vector) by default; the linkage editor being used automatically creates an object structure listing or external symbol dictionary from it, and this is loaded along with the program when standard options are used (see section 4.1 in the AID Core Manual [1]). This provides AID with the required information on CSECTs and COMMONs.

If the generation of this information was explicitly suppressed when linking the program with TSOSLNK (SYMTEST=NO) or BINDER (SYMBOL-DICTIONARY=NO operand in the SAVE-LLM statement) or if the information was not loaded, the following functions cannot be executed:

- output of a list of all CSECTs of the user program (%D %SORTEDMAP or %D %MAP)
- output of machine-oriented localization information (%D %LOC (memref))
- specification of a CSECT/COM qualification in a memory reference
- monitoring program execution via %CONTROLn and %TRACE commands implicitly or explicitly restricted to one CSECT/COMMON
- specification of CSECT-relative information for %DISASSEMBLE, %FIND, %TRACE, and in the STOP message
- generation of REPs for modifications

CSECTs that have been renamed must be addressed in AID commands by their new names (see chapter chapter 4 in the AID Core Manual [1]).

Be careful with LLMs or contexts which include CSECTs of the same name; it is not possible to predict which CSECT is addressed by AID in such cases.

If you are loading your program with ELDE or linking and loading it with DBL (DLL until BS2000 V9.5), no operands need to be specified.

3.2 Commands at the beginning of a debugging session

AID commands do **not** have an SDF syntax. This means that:

– operands are not requested via menus;

– an error message is issued in the case of an error, but no correction dialog is executed.

If your task is currently not in SDF-EXPERT mode, you will need to switch to EXPERT mode with the command `MODIFY-SDF-OPTIONS GUIDANCE=EXPERT` in order to enter AID commands.

The following option must be set with the `/MODIFY-TERMINAL-OPTIONS` command to enable the interruption of extensive AID outputs with the K2 key:
`OVERFLOW-CONTROL=USER-ACKNOWLEDGE`

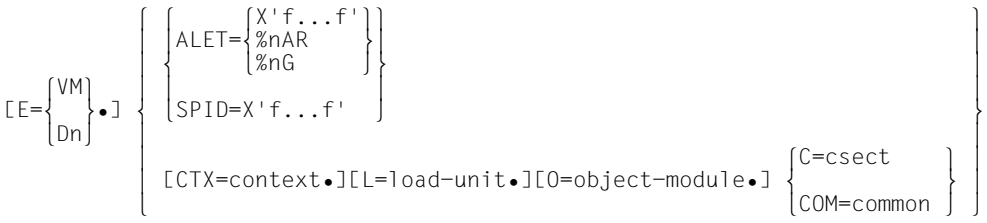
4 Machine-code-specific addressing

This chapter describes the qualifications and memory references that you can use for debugging on machine code level. A general description of addressing can be found in chapter 7 of the AID Core Manual [1].
If you specify a memory object that is not in the current AID work area or cannot be uniquely addressed in it, you must precede the memory reference with the qualifications that will allow AID to make the assignment. The operations described in chapter 7 of the AID Core Manual [1] can be applied to all memory references.

If you are debugging programs with ESA instructions on ESA systems running a BS2000/OSD-BC version ≥ V1.0, you can use the ALET and SPID qualifications. These two qualifications allow you to reference virtual addresses in a data area. For more information on the concept of memory on ESA systems see the “Executive Macros” manual [7].

4.1 Qualifications

When debugging on machine code level, you can use the base qualification, and the CTX, L, O, C, COM, ALET, and SPID qualifications as area qualifications.
These qualifications must always be specified in the order in which they are described here. The individual qualifications are separated by periods. A period must also be entered between the last qualification and the following address. The overview below shows you how qualifications are used:



Base qualification

The base qualification is used to define the AID work area. You specify whether an adjacent address is to be located in virtual memory or in a dump file.

E={VM | Dn}

Defines whether AID is to access virtual memory (VM) or a dump file (Dn). The E qualification is described in chapter 7 of the AID Core Manual [1] and under the %BASE command. It may be immediately followed by a virtual address, a keyword or a complex memory reference.

Area qualifications

An area qualification designates a contiguous subsection of a program. If an address operand ends with one of these qualifications, it restricts the effect of a command to that area.

CTX=context

Designates a context (see section 7.1 in the AID Core Manual [1]). This qualification precedes an L, O, C, or COM qualification. An address operand can end with the CTX qualification only in the %QUALIFY command. This qualification is required in order to reference a translation unit, a CSECT or a COMMON which is contained in multiple contexts but is not the location of the current interrupt point.
context may be the context name that was explicitly assigned in the BIND macro or the implicitly assigned name LOCAL#DEFAULT or CTXPHASE. Programs loaded with DBL are also assigned the standard context name LOCAL#DEFAULT. If they were linked using TSOSLNK, the context is called CTXPHASE. Additional contexts for a program may be created as a result of a link to a shared-code program.
context must be a context name not exceeding 32 position.

[L=load-unit•] [O=object-module•] {C=csect | COM=common}

The L and/or O qualification is required only if you need to address a CSECT or COMMON that is not the current one and only if the current context includes multiple CSECTs/ COMMONs with the same name. You only need to specify the L and/or O qualifications required for a unique address. A C/COM qualification must follow. Information on L, O, and C/COM qualifications for a virtual address can be obtained using %DISPLAY %LOC.

L=load-unit

Designates all link and load modules or a single load module that must be located in virtual memory or the corresponding dump at the time it is addressed.

load-unit Name of an object module (OM); ≤ 8 characters
Name of a link and load module (LLM); ≤ 32 characters.

O=object-module

Designates an object module (OM) as entered during compilation. The *load-unit* containing the object module must be located in virtual memory or in the corresponding dump at the time the object module is addressed.

object-module Name of an object module; ≤ 8 characters.

C=csect

Designates a CSECT. The CSECT name was assigned in the source program, but may have been modified with LMS, TSOSLNK or BINDER (see chapter 4 of the AID Core Manual [1]). *csect* must always be the last declared name. You may also specify a masked CSECT; however, you cannot use CSECT names for addressing if no object structure list was created by the linkage editor.

csect Name of a CSECT; ≤ 32 characters.

If the name contains no special characters, you must enter *csect* immediately after *C=*; otherwise, it must be placed within quotes in the form *C=N'...'*.

AID displays the names of the CSECTs in a test object in the output of %DISPLAY %SORTEDMAP or %DISPLAY %MAP.

An address operand may end with a C qualification. This designates the entire CSECT in the %DISPLAY, %CONTROLn, %FIND, %MOVE, %SET and %TRACE commands.

In the %DISASSEMBLE and %INSERT commands, it designates the start address of the CSECT.

The C qualification can also be used as a memory reference, since it has both an address attribute and a length attribute (see also the section on “Memory references”).

Be careful with LLMs or contexts which include CSECTs of the same name; it is not possible to predict which CSECT is addressed by AID in such cases.

COM=common

Designates a COMMON. Since the COM qualification can be used just like the C qualification, refer to that description for details.

ALET={X'f...f' | %nAR | %nG}

Designates a data area via one of its ALETs. The ALET qualification is only required when debugging ESA programs. The ALET is returned by the ALESRV macro. It can be specified as a hexadecimal literal or be taken from an access register or an AID register.
An ALET qualification may be followed by a virtual address or a complex memory reference without symbolic components.

X'f...f' is an 8-character hexadecimal literal.
%nAR is an access register; $0 \leq n \leq 15$
%nG is an AID register in which the ALET was buffered; $0 \leq n \leq 15$

SPID=X'f...f'

Designates a data area via its SPID, an identifier 8 bytes in length. The SPID qualification is only required when debugging ESA programs. It is returned by the macro DSPSRV and can only be specified as a hexadecimal literal.
An SPID qualification may only be followed by a virtual address or a complex memory reference without symbolic components.
X'f...f' is a 16-character hexadecimal literal.

You can obtain information on the ALET/SPID qualifications by entering the following command:
%DISPLAY %DS[(ALET/SPID-qua)]

4.2 Memory references

In debugging on machine code level, a memory location is referred to by a virtual address, a complex memory reference, a C/COM qualification, or a keyword.
If a program (segment) for which LSD records have been generated is being debugged, the data names and statement names defined in the source program can also be used in all operands in which *compl-memref* is possible.

V'f...f' Designates a virtual address. It has the implicit length 4, the storage type %X, and thus the output type dump.
f...f is a hexadecimal address of up to 8 digits, where *f* may be any value from 0 - 9 and A - F.
If you wish to reference an address in a data area when debugging ESA programs, you must specify an ALET or SPID qualification before the virtual address. Otherwise, a virtual address is unique in the respective AID work area, and only the specification of a base qualification is meaningful. Any area qualifications that are specified are thus redundant and ignored by AID.

{C=csect | COM=common}

Designates a CSECT or a COMMON that has all attributes of a virtual address and also the name attribute. The C/COM qualification can therefore also be used as a start address of a *compl-memref*. The area limits of the CSECT/COMMON can only be exceeded by indirect addressing (->). More detailed information can be found under “Area qualifications” on page 11.

keyword

Designates a storage class, program register, AID register, system information, program counters or the addressing mode, depending on the operand type (see chapter 10 in the AID Core Manual [1]). Each keyword has all the attributes of a memory object (see section 7.2.3 in the AID Core Manual [1]). The area limits of a keyword can only be exceeded by indirect addressing (->).

compl-memref

A complex memory reference designates a calculated address. Without the appropriate type and length modification, it has the implicit length 4, the storage type %X, and the output type dump.
The following operations may occur in *compl-memref* when debugging programs **not** running on ESA systems (see chapter 7 of the AID Core Manual [1]):

– byte offset (•)
– indirect addressing (->)
– type modification (%T(dataname), %X, %C, %P, %D, %F, %A, %S, %SX)
– length modification (%L(...), %L=(expression), %Ln)
– address selection (%@(...))

After a byte offset or indirect addressing, the implicit storage type and length of the start address are lost, and %XL4 applies. However, the area limits of the start address (e.g. CSECT or keyword) remain in effect. They must not be exceeded with any operand in a *compl-memref* (byte offset, length or type modification); otherwise, AID will issue an error message. It is only by connecting the address selector with the pointer operator or indirect addressing that a switch is made to the machine code level, i.e. the level on which the area includes the complete virtual memory occupied by the loaded program.

If you are debugging programs on ESA systems, you should note the following differences in connection with operations that may occur in *compl-memref*:

– You can mix symbolic and machine-oriented addressing for memory objects in the **program area**; the available options are the same as those for programs on non-ESA systems.

– You **cannot** use **symbolic** addressing for memory objects in the **data area**; **only** machine-oriented components may be used. The ALET or SPID qualifications have an effect **on all** components of the complex memory reference. A name can only be used in a following type modification. Without a preceding area qualification, AID extends only the current base qualification.

```
%DISPLAY ALET=%2G.V‘34‘%T(NAME)
```
corresponds to
```
%DISPLAY E=VM.ALET=%2G.V‘34‘ %T(E=VM.NAME)
```

Examples

Addressing and special operands when debugging ESA programs:

1.
```
%DISPLAY %ASC
%ASC            = 01
```

The keyword %ASC informs you if access registers were used in the address conversion and if data areas were addressed with them. The value 01 indicates that access registers were used.

2.
```
%DISPLAY %DS
SPID                ALET       SIZE

0001003300000031    00000000   0
0000000080000800    0001001B   409600
0000000080000800    0001001C   409600
0000000080000900    00010011   131072
0000000080000900    00010012   131072
```

All entries for the active data areas are output.

3.
```
%DISPLAY %AR
%AR   ( 0 : 15)
(  0) 00000000  (  1) 000000FF  (  2) 00000000  (  3) 00000090  (  4) 00000094
(  5) 000000B4  (  6) 92020711  (  7) 09252742  (  8) 00000000  (  9) 00000000
( 10) 000000AC  ( 11) 000000B0  ( 12) 000000B4  ( 13) 92020711  ( 14) 09252742
( 15) 00000000
```

All access registers are output. The access registers with a value < 0 contain an ALET.

4. ```
   %FIND C‘01‘ IN ALET=X‘00010003‘.V‘0‘%L100
   ABSOLUT +0000000F=0000000F : F0F1405E2 C5C9E3C5 40F0F0F0  01 PAGE 000
   %FIND C‘01‘ IN ALET=%8AR.V‘0‘%L100
   ABSOLUT +0000000F=0000000F : F0F1405E2 C5C9E3C5 40F0F0F0  01 PAGE 000
   %FIND C‘01‘ IN SPID=X‘0000000080000200‘.V‘0‘%L100
   ABSOLUT +0000000F=0000000F : F0F1405E2 C5C9E3C5 40F0F0F0  01 PAGE 000
   ```

   Three methods of addressing the same location in the data area are shown.
   ```

5 AID commands

%AID

The %AID command can be used to declare global settings or to revoke the settings valid up until then.

– By means of the *CHECK* operand you define whether an update dialog is to be initiated prior to execution of a %MOVE or %SET command.

– By means of the *REP* operand you define whether memory updates of a %MOVE command are to be stored as REPs.

– By means of the *SYMCHARS* operand you define whether AID is to interpret a "-" in program, data and statement names as a hyphen or as a minus sign.

– By means of the *OV* operand you can direct AID to take the overlay structure of a program into account.

– By means of the *LOW* operand you direct AID to convert lowercase letters of character literals and names to uppercase, or to interpret them as lowercase. OFF is the default.

– By means of the *DELIM* operand you define the delimiters for AID output of alphanumeric data. The vertical bar is the default delimiter.

– By means of the *LANG* operand you define whether AID is to output %HELP information in English or German.

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

%AID can only be issued as an individual command, it must never be part of a command sequence or a subcommand.

The %AID command does not alter the program state.

CHECK

ALL Prior to execution of a %MOVE or %SET command, AID conducts the following update dialog:

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

N

AID0342 NOTHING CHANGED
```

If **Y** is entered, the old contents of the data field are overwritten and no further message is issued. In procedures in batch mode AID is not able to conduct a dialog and always assumes **Y**
The old and new contents are written to SYSOUT. If SYSOUT is redirected, the above output will not appear on the terminal. The same applies if the %MOVE or %SET command and the CMD macro have been used and output to SYSOUT has been selected. Messages AID0274 and AID0342, by contrast, are always sent to the terminal.

NO %MOVE and %SET commands are executed without an update dialog.

If the *CHECK* operand is entered without specification of a value, AID assumes the default value (NO).

REP

YES In the event of memory updates caused by a %MOVE command, LMS correction statements (REPs) are created in SDF format. If the object structure list is not available, AID does not create any REPs and issues an error message to this effect.

AID stores the corrections in a file with the link name F6. The MODIFY-ELEMENT statement must then be inserted in it for the LMS run. Care should therefore be taken that no other outputs are written to the file with link name F6.

If no file with link name F6 is registered (see %OUTFILE), the REP record is stored in the file AID.OUTFILE.F6 created by AID.
User-specific REP files must be created with FCBTYPE=SAM. REP files created by

AID are likewise defined with FCBTYPE=SAM, RECFORM=V and OPEN=EXTEND. The file remains open until it is closed with an %OUTFILE command or until /LOGOFF or /EXIT-JOB.

NO No REPs are generated.

If the *REP* operand is entered without a value specification, AID inserts the default (NO). The *REP* operand of the %MOVE command can supersede the declaration made with %AID, but only for this particular %MOVE command. For subsequent %MOVE commands without a *REP* operand, the declaration made with the %AID command is valid again.

SYMCHARS

STD A hyphen "-" is interpreted as an alphanumeric character and can, as such, be used in program, data and statement names. It is interpreted as a minus sign, however, if a blank precedes it.

NOSTD
A hyphen "-" is always interpreted as a minus sign and cannot be part of names.

If the *SYMCHARS* operand is entered without a value specification, AID inserts the default value (STD).

OV

YES Mandatory specification if the user is debugging a program with an overlay structure. AID checks each time whether the program unit which has been addressed originates from a dynamically loaded segment.

NO AID assumes that the program to be debugged has been linked without an overlay structure. AID uses the loaded LSD records without checking whether the program unit addressed might be situated in a dynamically loaded segment.

If the *OV* operand is entered without a value specification, AID assumes the default (NO).

LOW

ON Lowercase letters in character literals and in program, data and statement names are not converted to uppercase.

OFF All lowercase letters from user entries are converted to uppercase.

If no *LOW* operand is entered during a debugging session, OFF applies. If the *LOW* operand is entered without a value specification, AID assumes the default (ON). To reactivate conversion to uppercase LOW=OFF must be entered.

DELIM

C'x'|'x'C|'x'

With this operand the user defines a character as the left-hand and right-hand delimiter for AID output of symbolic character-type data with the %DISPLAY command.

| The standard delimiter is the vertical bar.

If the *DELIM* operand is entered without value specification, AID inserts the default value (|).

LANG

D AID outputs information requested with %HELP in German.

E AID outputs information requested with %HELP in English.

If the *LANG* operand is entered without a value, AID inserts the default (D).
You can also receive AID messages in German by using the SDFcommand
`MODIFY-MSG-ATTRIBUTES TASK-LANGUAGE=D`.
This has no effect on the update dialog (see the CHECK operand).

%AINT

%AINT is used to define whether indirect addressing with AID is to work with 24-bit or 31-bit addresses. For AID, the address preceding the pointer operator (->) then consists of 24 or 31 bits accordingly.

The addressing mode of the test object is not affected.

– By means of *aid-mode* you define address interpretation for indirect addressing within an AID work area.

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

%AINT is only meaningful if the program to be tested was generated on an XS processor.

As a standard procedure, AID interprets indirect address specifications in accordance with the current addressing mode of the test object. The current addressing mode is the one which appears at an interrupt point in the system information field %AMODE. As long as an %AINT command is valid, no matching to the current addressing mode takes place and indirect addresses are interpreted in accordance with the declarations made under %AINT. The current addressing mode (%AMODE) of the test object can be queried using the %DISPLAY command and modified using the %MOVE command.

If no qualification has been specified, %AINT is valid for AID commands which reference or use indirect addresses of the current AID work area.

%AINT without operands switches back to standard address interpretation. The same effect is achieved by an %AINT command with a base qualification and without any specification pertaining to address interpretation.

The %AINT command does not alter the program state.

aid-mode

Defines how indirect addresses in subsequent AID commands are to be interpreted for the current AID work area or the one designated by the base qualification specified.

If a keyword is specified for address interpretation and no qualification, the %AINT command is valid for processing of the current AID work area.

If a base qualification is specified but no keyword for address interpretation, the AID standard address interpretation goes into effect for the corresponding AID work area.

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

                 ⎧VM⎫          ⎧%M[ODE]31⎫
[•][E=⎨  ⎬[•]] [⎨         ⎬]
                 ⎩Dn⎭          ⎩%M[ODE]24⎭

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

- If the period is in a leading position, it identifies a *prequalification*, which must have been defined in a preceding %QUALIFY command.
  A period must be inserted between the base qualification and the keyword for address interpretation. If only a base qualification is specified, no final period is permitted.

E={VM | Dn}
: is specified if switching of address interpretation is not to apply for the current AID work area. If only a base qualification is specified, the standard address interpretation applies again for the area addressed.

{%M[ODE]31 | %M[ODE]24}
: Keyword specifying the number of bits to be taken into account for indirect addressing in AID commands.

```
%M[ODE]31   31-bit addressing
%M[ODE]24   24-bit addressing
```

Examples

The contents of address V’100’ are: 1200000C

The contents of register 5 are: 010001A0

1. ```
   %AINT %MODE24
   %DISPLAY V'100'->
   %MOVE %5-> INTO %5G
   ```
   %AINT is used to switch over to 24-bit address interpretation. The changeover is valid for the current AID work area.
   The %DISPLAY command outputs four bytes as of address V’00000C’.
   The %MOVE command transfers four bytes as of address V’0001A0’ to AID register 5.

2. ```
   %AINT %MODE31
   %DISPLAY V'100'->
   %MOVE %5-> INTO %5G
   ```
   The user switches address interpretation for the current AID work area to 31-bit interpretation.
   The %DISPLAY command outputs four bytes as of address V’1200000C’.
   The %MOVE command transfers four bytes as of address V’010001A0’ to AID register 5.

%BASE

The %BASE command is used to specify the base qualification. All subsequently entered memory references without a new base qualification assume the values declared by %BASE. The command also defines where the AID work area is to be located.

– With the *base* operand the user designates either the virtual memory area of the program which has been loaded or a dump in a dump file.

Command	Operand
%BASE	[base]

When debugging on machine code level, the AID work area corresponds to the non-privileged area in virtual memory that is occupied by the loaded program and all connected subsystems or to the corresponding area in a dump. If you do not specify a %BASE command during a debugging session or enter %BASE without any operands, the base qualification E=VM applies by default.

A %BASE command remains in effect until the next %BASE command or a /LOGOFF or /EXIT-JOB, is issued, or until the dump file that was declared as the base qualification is closed (see %DUMPFILE).

Immediately after input, all memory references in a command, even within a subcommand, are supplemented with the current base qualification, i.e. a %BASE command has no effect on subcommands specified previously.

%BASE can only be entered as an individual command, it must never be part of a command sequence or subcommand.

The %BASE command does not alter the status of the program.

base

Defines the base qualification. All subsequently entered memory references without a separate base qualification assume the value declared with the %BASE command.

base–OPERAND

E = { VM | Dn }

E=VM

The virtual memory area of the program which has been loaded is declared as the base qualification. VM is the default value, thus virtual memory is also the AID standard work area.

E=Dn

A dump in a dump file with the link name *Dn* is declared as the base qualification. *n* is a number with a value $0 \leq n \leq 7$.

Before declaring a dump file as the base qualification, the user must assign the corresponding dump file to a link name and open it, using the %DUMPFILE command.

%CONTINUE

The %CONTINUE command is used to start the program which has been loaded or to continue it at the interrupt point.
As opposed to %RESUME, a %CONTINUE command does not terminate an interrupted %TRACE but continues it depending on the declarations which have been made.

Command	Operand
%CONT[INUE]	

In the following cases a %TRACE command is regarded as interrupted and is resumed by any %CONTINUE command:

1. When a subcommand has been executed as the result of a monitoring condition from a %CONTROLn, %INSERT or %ON command having been satisfied, and the subcommand contained a %STOP.
2. When an %INSERT command terminates with a program interrupt because the *control* operand is K or S.
3. When the K2 key has been pressed (see section 3.2, “Commands at the beginning of a debugging session” on page 8).

If the %CONTINUE command is the only one in a subcommand, only the execution counter is incremented.

If the %CONTINUE command is given in a command sequence or subcommand, any subsequent commands are not executed.

%CONTINUE alters the program state.

%CONTROLn

By means of the %CONTROLn command you may declare up to seven monitoring functions one after the other, which then go into effect simultaneously. The seven commands are %CONTROL1 through %CONTROL7.

– By means of the *criterion* operand you may select different types of machine instructions. If an instruction of the selected type is about to be executed, AID interrupts the program and processes *subcmd*.

– By means of the *control-area* operand you may define the program area in which *subcmd* is to be processed if *criterion* is satisfied.

– By means of the *subcmd* operand you declare a command or a command sequence and possibly a condition. *subcmd* is executed if *criterion* is satisfied and any specified condition has been met.
Specification of the *subcmd* operand is mandatory.

Command	Operand		
%C[ONTROL]n	criterion	[IN control-area]	<subcmd>

A %CONTROL*n* on machine code level **cannot** be active **at the same time** as a *write-event* of %ON.

Several %CONTROL*n* commands with different numbers do not affect one another. Therefore you may activate several commands with the same *criterion* for different areas, or with different *criteria* for the same area. If several %CONTROL*n* commands occur for one machine instruction, the associated subcommands are executed successively, starting with %C1 and working through %C7.

The **effect** of %CONTROL*n* when debugging on machine code level **is different** from debugging on the symbolic level.
When debugging on machine code level, *criterion* is **also monitored outside** the *control-area*, but *subcmd* is only executed within the *control-area*. The additional monitoring overhead slows down execution of the program. In large programs, the %CONTROL*n* command should therefore be used directly at test points as a subcommand of the %INSERT command and deleted after execution (see %INSERT and chapter 6 in the AID Core Manual [1]).

The individual value of an operand for %CONTROL*n* is valid until overwritten by a new specification in a later %CONTROLn command with the same number, until the %CONTROL*n* command is deleted or until the end of the program.

%REMOVE can be used to delete an individual %CONTROL*n* command or all active %CONTROL*n* declarations.

For %CONTROL*n* the base qualification E=VM must be valid (see %BASE) or specified explicitly.

The %CONTROLn command does not alter the program state.

criterion

Keyword defining the type of machine instructions prior to whose execution AID is to process *subcmd*.

Since the **default value** for *criterion* is the **symbolic** %STMT, *criterion* must **always** be specified for machine-oriented debugging if no *criterion* declaration from a previous %CONTROL*n* command is still valid.

criterion	*subcmd* **is processed** prior to**:**
%INSTR	every machine instruction pending execution
%B	every branch instruction pending execution (i.e. the machine instructions BAL, BALR, BAS, BASSM, BASR, BC, BCR, BCT, BCTR, BSM, BXH and BXLE)
%BAL	every subprogram call pending (as the result of machine instructions BAL, BALR, BAS, BASSM and BASR)

control-area

Defines the program area in which the monitoring functionhas an effect. When the specified program area is exited, monitoring of *criterion* continues, but *subcmd* is no longer processed. When a machine instruction in the program area to be monitored is pending execution, *subcmd* is processed again.

A *control-area* definition is valid until the next %CONTROLn command with the same number is issued with a new definition, until the corresponding %REMOVE %CONTROLn is given, or until the end of the program. %CONTROLn without a *control-area* operand of its own results in a valid area definition being taken over. To be valid, *control-area* must be in defined in a %CONTROLn command with the same number, and the current interrupt point must be within this area. If no valid area definition exists, the *control-area* comprises the entire user program by default.

```
control-area-OPERAND - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

                     ⎧ C=csect | COM=common   ⎫
IN [•][qua•] [•] ⎨ ( V'f...f': V'f...f' ) ⎬
                     ⎩ keyword                ⎭

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

- If the period is in the leading position, it denotes a *prequalification* which must have been defined with a preceding %QUALIFY command. Consecutive qualifications must be separated by a period. In addition, there must be a period between the final qualification and the following operand part.

qua One or more qualifications may be specified here if a memory object to be addressed is not located in the current AID work area or is not unique in that area. It is sufficient to specify only the qualifications needed for a unique address.

E=VM
: As *control-area* can only be located in the virtual memory of the loaded program, the specification *E=VM* is required if a dump file has been declared as the base qualification.

CTX=context
: Specified only to enable unique addressing of a CSECT or a COMMON in the program system.

[L=load-unit•][O=object-module•]
: Specified only if a CSECT or COMMON to be referenced cannot be uniquely addressed in the current context without these qualifications. You only need to specify the qualifications required for a unique address.

{C=csect | COM=common}
: The *control-area* covers the entire CSECT or COMMON specified.

(V'f...f':V'f...f')
: The *control-area* is defined by specifying a virtual start address and a virtual end address.
The addresses must be located in the executable part of a loaded program.
The start address must be less than or equal to the end address.

keyword

The *control-area* covers the memory area addressed by one of the following keywords (see chapter 9 in the AID Core Manual [1]).
In the case of XS and ESA systems, %CLASS6 and %CLASS6BELOW designate the same address space.

```
%CLASS6           Class 6 memory
%CLASS6BELOW      Class 6 memory below the 16-Mb limit
%CLASS6ABOVE      Class 6 memory above the 16-Mb limit
```

subcmd

This is processed whenever a machine instruction corresponding to *criterion* is pending in *control-area*. The *subcmd* is executed before the instruction. Specification of *subcmd* is mandatory, as AID inserts no <%STOP> for %CONTROLn.

A complete description of *subcmd* can be found in chapter 6 of the AID Core Manual [1].

```
subcmd-OPERAND  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

                                      ⎧AID-command    ⎫
< [subcmdname:] [(condition):] [      ⎨               ⎬ {;...}] >
                                      ⎩BS2000-command ⎭

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

The subcommand may contain a name, a condition and a command part. This command portion can consist of an individual command or a command sequence; it may contain AID commands, BS2000 commands and/or comments. There is an execution counter for each subcommand. For information on how to formulate an execution condition, assign names and execution counters and address them, and which commands are not permitted within subcommands, refer to the AID Core Manual, chapter 5.

If a subcommand consists of only a name or only a condition, i.e. if the command part is missing, AID merely increments the execution counter when the instruction selected with *criterion* is executed.

In addition to the commands which are not permitted in any subcommand, the *subcmd* of a %CONTROLn must not contain the AID commands %CONTROLn, %INSERT, %JUMP or %ON.

The commands in *subcmd* are executed consecutively, after which the program is continued. The commands for runtime control also immediately change the program state when they are part of a subcommand. They abort *subcmd* and continue the program (%CONTINUE, %RESUME, %TRACE) or halt it (%STOP). In practice, they are only useful as the last command in *subcmd*, since any subsequent commands of the *subcmd* will not be executed. This also applies to the deletion of an active subcommand with %REMOVE.

Examples

1. `%C1 %INSTR <C1_COUNTER: %CONT>`
   Prior to each instruction, the counter of subcommand C1_COUNTER is incremented by one, after which the program continues.

2. `%C1 %B IN C=M1BS <%D %PC–>%XL8>`
   In CSECT M1BS, eight bytes of the program code are output at the interrupt point for each branch instruction (%PC->). After this the program is continued and any active %TRACE will be resumed. Program execution is, however, also monitored outside M1BS and is therefore considerably slowed down.

3. `%INSERT V'B4O' <%REM %C5>`
   `%INSERT V'A38' <%C5 %B IN (V'A38':V'B4O') <%D V'798'>>`
   From address V’A38’ to V’B40’, four bytes are to be output for each branch instruction, starting at address V’798’. The command does not, however, become active until the program run reaches address V’A38’ (%INSERT), and the %CONTROL5 is deleted again at the end of the area to be monitored. Program execution outside the area is thus not slowed down.

4. `%C2 %BAL <(%PC–>%L2 EQ X'O5EF'): %D %PC, %15>`
   All BALR 14,15 instructions are logged with both the exit point (%PC) and the target address (%15).

%DISASSEMBLE

%DISASSEMBLE enables memory contents to be "retranslated" into symbolic Assembler notation and output.

– The *number* operand enables you to determine how many output lines with interpreted memory contents are to be output.

– The *start* operand enables you to determine the address where AID is to begin disassembling.

Command	Operand
{%DISASSEMBLE / %DA}	[number] [FROM start]

Disassembly of the memory contents starts with the first byte. For memory contents which cannot be interpreted as an instruction, an output line is generated which contains the hexadecimal representation of the memory contents and the message INVALID OPCODE. The search for a valid operation code then proceeds in steps of 2 bytes each.

%DISASSEMBLE without a *start* operand permits the user to continue a previously issued %DISASSEMBLE command until the test object is switched by means of a BS2000 or AID command (/START-PROGRAM, /LOAD-PROGRAM, %BASE) or a new operand value is declared. AID continues disassembly at the memory address following the address last processed by the previous %DISASSEMBLE command. If *number* is not specified either, AID generates the same number of output lines as declared before.

If the user has not entered a %DISASSEMBLE command during a test session or has switched the test object and does not specify current values for one or both operands in the %DISASSEMBLE command, AID works with the default value 10 for *number* and V'0' for *start*. If the program was not loaded starting at V'0', *start* must be specified.

The %OUT command can be used to control how processed memory information is to be represented and whether it is to be output to SYSOUT, SYSLST or a catalog file. The format of the output lines is explained after the description of the *start* operand.

The %DISASSEMBLE command does not alter the program state.

number

Specifies how many Assembler instructions are to be output.

If no value has been specified for *number* and no value from a previous %DISASSEMBLE command applies, AID inserts the default value (10).

number
: is an integer with the value:
$1 \leq number \leq 2^{31}-1$

start

Defines the address at which disassembly of memory contents into Assembler instructions is to begin. If the *start* value is not specified, AID inserts the address V'0' for the first %DA. For every further %DA, disassembly is continued after the last Assembler instruction that was disassembled.

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

                ⎧ C=csect | COM=common ⎫
                ⎪ V'f...f'             ⎪
FROM [•][qua•]⎨                        ⎬
                ⎪ keyword              ⎪
                ⎩ compl-memref         ⎭

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

- If the period is in the leading position it denotes a *prequalification*, which must have been defined by a previous %QUALIFY command.
  Consecutive qualifications must be delimited by a period. In addition, there must be a period between the final qualification and the following operand part.

qua
: One or more qualifications may be specified here if a memory object to be addressed is not located in the current AID work area or is not unique in that area. It is sufficient to specify only the qualifications needed for a unique address.

  E={VM | Dn}
  : Only required if the current base qualification does not apply to *start* (see %BASE).

  CTX=context
  : Specified only to enable unique addressing of a CSECT or a COMMON in the program system.

[L=load-unit•][O=object-module•]
Specified only if a CSECT or COMMON to be referenced cannot be uniquely addressed in the current context without these qualifications. You only need to specify the qualifications required for a unique address.

{C=csect | COM=common}
Defines *start* as the start address of the specified CSECT or COMMON.

V'f...f' is a virtual address that must lie in the executable segment of the program and must be the start address of a machine instruction. Otherwise, the disassembly will be meaningless.

keyword
Defines *start* as the start address of a memory class (see section 10.5 in the AID Core Manual [1]). In the case of XS and ESA systems, %CLASS6 and %CLASS6BELOW designate the same address area.

```
%CLASS6            Class 6 memory
%CLASS6BELOW       Class 6 memory below the 16-Mb limit
%CLASS6ABOVE       Class 6 memory above the 16-Mb limit
```

compl-memref
compl-memref should provide a start address of a machine instruction; otherwise, the result of disassembly will be meaningless. The following operations may occur in
compl-memref (see section 7.2.4 in the AID Core Manual [1] and chapter 4, "Machine-code-specific addressing" in this manual for restrictions on debugging ESA programs):

- byte offset (•)
- indirect addressing (->)
- type modification (%A, %S, %SX)
- length modification (%L(...), %L=(expression), %Ln)
- address selection (%@(...))

With indirect addressing (->) the user can employ the address stored in the program counter or in a register as *start*.
Example: %PC-> sets *start* to the interrupt point.

Output of the %DISASSEMBLE listing

By default, the %DISASSEMBLE listing is output with additional information to SYSOUT (T=MAX). With %OUT the user can select the output media and specify whether or not additional information is to be output by AID.

The following is contained in a %DA output line if the default value T=MAX is set:

– CSECT-relative memory address
– memory contents converted to symbolic Assembler notation, displacements being represented as hexadecimal numbers (in contrast to Assembler format)
– for memory contents which do not begin with a valid operation code, Assembler instruction DC in hexadecimal format and with a length of 2 bytes, followed by the note INVALID OPCODE
– hexadecimal representation of the memory contents (machine code).

The CSECT name is placed in a separate line if it exceeds 8 characters in length.
The next line then begins with the CSECT name, truncated to 7 characters and followed by an asterisk (*****), and otherwise corresponds to the structure described above.

Example of line format (T=MAX) for a CSECT with a short name

```
MOBS+A30        LM    R01,R1,C0(R11)          98 01 B0C0
MOBS+A34        L     R15,B4(R0,R11)          58 F0 B0B4
MOBS+A38        BALR  R14,R15                 05 EF
MOBS+A3A        IDL   5F0(R9),X'00'           80 00 95F0
MOBS+A3E        LPDR  R2,R0                   20 20
```

Example of line format (T=MAX) for a CSECT with a long name

```
                BCL45678901234567890&@
BCL4567*+62       CLC   105(6,R2),11E(R2)     D5 05 2105 211E
```

The %OUT operand value T=MIN produces abbreviated output lines, in which the CSECT-relative address is replaced by the virtual address and the hexadecimal representation of the memory contents is not included.

Example of line format (T=MIN)

```
00000A30        LM    R01,R1,C0(R11)
00000A34        L     R15,B4(R0,R11)
00000A38        BALR  R14,R15
00000A3A        IDL   5F0(R9),X'00'
00000A3E        LPDR  R2,R0
```

Examples

1. `%DISASSEMBLE 15 FROM CTX=CTXPHASE.V'A18'`
   The memory contents starting with virtual address 'A18' in the context CTXPHASE are to be disassembled into Assembler instructions. 15 output lines with valid or invalid operation code are generated. Memory contents containing no permissible operation code are output as a hexadecimal string with a length of two bytes and the note INVALID OPCODE.

2. `%DA 1 FROM %PC->`
   One instruction is disassembled beginning at the interrupt point.

3. `%DA FROM C=PRINT.8`
   Disassembly begins with the start address of control section PRINT+8. If no previous declaration for *number* exists, 10 output lines are generated.

4. `%QUALIFY L=RESI.O=OSS1.C=HEADER`
   `%DA 5 FROM .#'A0'->`
   AID supplements the command as follows:
   `%DA 5 FROM L=RESI.O=OSS1.C=HEADER.#'A0'->`
   #'A0' is added to the start address of control section HEADER in object module OSS1, which in turn is contained in load module RESI.
   This calculated address contains the address at which AID is to start disassembly.

5. `%OUT %DISASSEMBLE T=MIN`
   `%DA 50 FROM V'128'`
   50 lines with disassembled memory contents are to be generated as of virtual address '128'. The previous %OUT command defined that the hexadecimal representation of memory contents and the CSECT-relative specification of the memory address are to be omitted.

6. `%FIND X'D5' ALIGN=2`
   `%DA 1 FROM %0G->`
   The start address of a CLC instruction is to be located. The %FIND command stores the hit address in AID register %0G. AID is to disassemble one instruction starting at this address.

7. `%A 1 FROM C=CS1.(%L(C=CS1)-4)`
   The last four bytes of CSECT CS1 are to be disassembled.

%DISPLAY

The %DISPLAY command is used to output memory contents, addresses, lengths, system information and AID literals and to control feed to SYSLST. Output is via SYSOUT, SYSLST or to a cataloged file.

– By means of the *data* operand you specify a memory location whose content, address or length is to be output, or you designate system information, define AID literals or control feed to SYSLST.

– By means of the *medium-a-quantity* operand you specify the output medium AID uses and whether or not additional information is to be output. This operand disables a declaration made via the %OUT command, but only for the current %DISPLAY command.

Command	Operand
%D[ISPLAY]	data {,...} [medium-a-quantity][,...]

A %DISPLAY command which does not have a qualification for *data* addresses *data* in the current AID work area.

When debugging ESA programs, the addresses located in a data area are marked in the output by an asterisk.

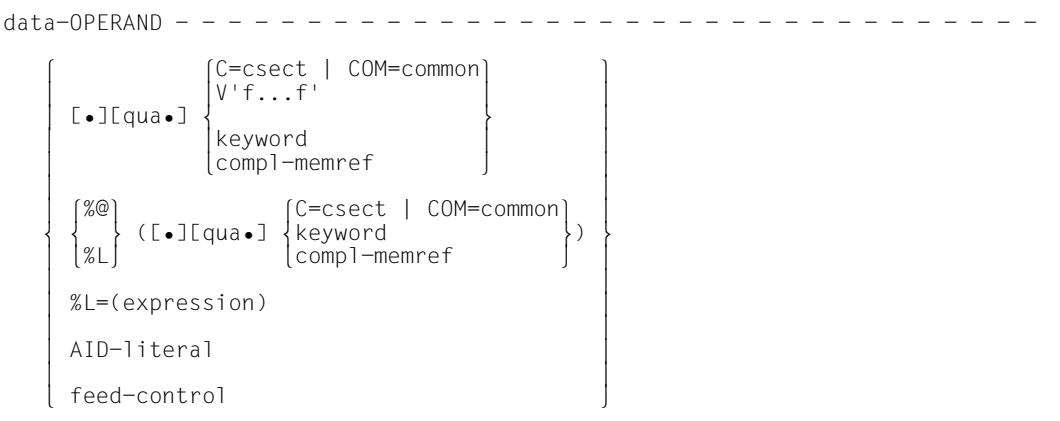
If the *medium-a-quantity* operand is not specified, AID outputs the data in accordance with the declarations in the %OUT command or, by default, to SYSOUT together with additional information.

The %DISPLAY command does not alter the program state.

data

This operand defines the information AID is to output. You may output memory contents, addresses, lengths and system information. AID literals can be defined to improve the readability of debugging logs, and feed to SYSLST can be controlled for the same purpose.

If you enter more than one *data* operand in a %DISPLAY command, you may switch from one operand to another between the non-symbolic entries described here and the symbolic entries described in the language-specific manuals [2] - [5].
Symbolic and machine-oriented specifications can also be combined within a complex memory reference. For symbolic specifications, however, the LSD records must also be available (see chapter 3 in the AID Core Manual [1]).



- If the period is in the leading position it denotes a *prequalification*, which must have been defined with a preceding %QUALIFY command.
  Consecutive qualifications must be separated by a period. In addition, there must be a period between the final *qualification* and the following operand part.

qua
: One or more qualifications may be specified here if a memory object to be addressed is not located in the current AID work area or is not unique in that area. It is sufficient to specify only the qualifications needed for a unique address.

  E={VM | Dn}
  : Only required if the current base qualification does not apply to *data*.

  {ALET={X'f...f' | %nAR | %nG} | SPID=X'f...f'}
  : Specified to reference an address in a data area, but only required when debugging ESA programs. These qualifications may only be followed by a V address or a *compl-memref*.

  CTX=context
  : Specified only to enable unique addressing of a CSECT or a COMMON in the program system.

  [L=load-unit•][O=object-module•]
  : Specified only if a CSECT or COMMON to be referenced cannot be uniquely addressed in the current context without these qualifications. You only need to specify the qualifications required for a unique address.

{C=csect | COM=common}
: If addressing ends with a C/COM qualification, the entire program segment that was selected is displayed.

V'f...f'

Designates a virtual address, where *f...f* is a valid hexadecimal address with up to eight digits. The contents at a virtual address are output in hexadecimal form with a length of four bytes by default (%XL4).

keyword

Here you may specify all the keywords for memory classes, program registers, AID registers and system information, as well as the keyword for the execution counter (see chapter 9 in the AID Core Manual [1]). Keywords returning more than one value are edited by AID in tabular form.

```
%CLASS5             Class 5 memory
%CLASS5BELOW        Class 5 memory below the 15-Mb limit
%CLASS5ABOVE        Class 5 memory above the 16-Mb limit
%CLASS6             Class 6 memory
%CLASS6BELOW        Class  6 memory below the 16-Mb limit
%CLASS6ABOVE        Class 6 memory above the 16-Mb limit

%n                  General register, 0 ≤ n ≤  15
%nD|E               Floating-point register, single/douple precision,
                    n = 0,2,4,6
%nQ                 Floating-point register, quadruple precision, n = 0,4
%nG                 AID general register, 0 ≤ n ≤ 15
%nGD                AID floating-point register, n = 0,2,4,6
%nAR                Access register, 0 ≤ n ≤ 15
%MR                 All 16 general registers in tabular form
%FR                 All 4 floating-point registers with double precision
                    edited in tabular form
%AR                 All 16 access registers in tabular form

%PC                 Program counter
%PM                 Program mask
%CC                 Condition code
%PCB                Process control block

%PCBLST             List of all process control blocks
%SORTEDMAP          A list of all CSECTs and COMMONs of the user
                    program and of the connected subsystems (sorted by
                    (name and address). Long names are truncated.
```

```
        ⎧ CTX=context [•L=load-unit]   ⎫
        ⎪ L=load-unit                  ⎪
%MAP [(⎨          ⎧USER⎫               ⎬)]
        ⎪ SCOPE = ⎨    ⎬               ⎪
        ⎩         ⎩ALL ⎭               ⎭
```

```
       {CTX=context|  L=load-unit}
                    If a path is specified, all CSECTs/COMMONs of the
                    specified context or the specified load unit are
                    listed.

       SCOPE=USER   The CSECTs/COMMONs of the default contexts CTXPHASE
                    and LOCAL#DEFAULT and of contexts generated with the
                    LNKCTX[@] operand of the BIND macro are listed.
```

```
          SCOPE=ALL  In addition to the CSECTs/COMMONs of the user-defined
                     contexts, a MAP of all contexts to which the program
                     has connected (e.g. DSSM subsystems or user pool
                     contexts) is output.
          All BLS names (context, load unit, CSECT and COMMON) are output
          in unabbreviated form. The output list is sorted by CSECT names
          within the contexts and load units.

     %AMODE            Addressing mode (24 or 31-bit addresses)
     %ASC              ASC mode on ESA systems
                       (with AR mode: X'00' = off; X'01' = on)
     %AUD1             P1 audit table, plus the SAVE table (if any)

     %LINK             Name of the last dynamically loaded module/segment
                       (see event %LPOV under %ON)

     %LOC(memref)      Localization information for a memory referencein the
                       executable part of a program

     %DS[(ALET/SPID-qua)]
                       Information on SPIDs and/or ALETs of the active
                       data areas on ESA systems.

     %•[subcmdname]    Execution counter. The abbreviation %. designates
                       the execution counter of the currently active
                       subcommand.
```

compl-memref

Designates a calculated address. The following operations may occur in *compl-memref* (see section 7.2.4 in the AID Core Manual [1] and chapter 4 of this manual for restrictions on debugging ESA programs):

- byte offset (•)
- indirect addressing (->)
- type modification (%T(dataname), %X, %C, %P, %D, %F, %A, %S, %SX)
- length modification (%L(...), %L=(expression), %Ln)
- address selection (%@(...))

Following byte offset or indirect addressing, AID outputs the memory contents at the calculated address in dump format (%XL4) by default. Using the type modification, *data* may be edited in a different format, since the type of output depends on the storage type.
With the length modification you can define the output length yourself, e.g. if you wish to output only parts of a CSECT or memory area or if you wish to output more than the four default bytes.

If LSD records exist, the data names and statement names defined in the source program may also be used within *compl-memref*.

%@(...)

Using the address selector you can output the start address of a CSECT or a memory area (%CLASS6, %CLASS6BELOW, %CLASS6ABOVE) or of *compl-memref*.
The address selector supplies an address constant which you can use in a complex memory reference before a pointer operator (->). See also section 7.2.4.4 in the AID Core Manual [1].

%L(...)

Using the length selector you can output the length of a CSECT or a memory area (%CLASS6, %CLASS6BELOW, %CLASS6ABOVE). The length selector supplies an integer value which you can use for the calculation of a byte offset or in a length function (see section 7.2.4.4 in the AID Core Manual [1]).
Example: %D %L(C=CS1) outputs the length of CSECT CS1.

%L=(expression)

With the length function you can calculate a value. *expression* is comprised of the contents of memory references, constants, integers and the arithmetic operators (+, –, *, /). Only memory reference contents of type %F or %A with a length less than or equal to 4 are permitted. As a result the length function produces an integer which can be used for byte offset calculation, in a further length function or for length modification (see section 7.2.4.4 in the AID Core Manual [1]).
Example: %D %L=(%1) outputs the contents of register 1 as a decimal number.

AID-literal

All AID literals described in the AID Core Manual, chapter 8, may be specified. They are used for output log layout and annotation.

AID-literal	Meaning
{C'x...x' | 'x...x'C | 'x...x'}	Character literal
{X'f...f' | 'f...f'X}	Hexadecimal literal
{B'b...b' | 'b...b'B}	Binary literal
[{±}]n	Integer
#'f...f'	Hexadecimal number
[{±}]n.m	Fixed-point number
[{±}]mantissaE[{±}]exponent	Floating-point number

feed-control

For output to SYSLST, print editing can be controlled by the following two keywords, where:
%NPresults in a page feed.
%NL[(n)]results in a line feed by *n* blank lines.
$1 \leq n \leq 255$. The default for *n* is 1.

medium-a-quantity

Defines the medium or media via which output is to take place, and whether additional information is to be output besides the contents of the specified memory areas.

If this operand is omitted and no declaration has been made using the %OUT command, AID uses the presetting T=MAX.

```
medium-a-quantity-OPERAND - - - - - - - - - - - - - - - - - - - - - - - - - -

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

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

medium-a-quantity is described in detail in chapter 8 of the AID Core Manual [1].

T	Terminal output
H	Hardcopy output (includes terminal output; cannot be specified in conjunction with *T*)
Fn	File output
P	Output to SYSLST
MAX	Output with additional information.
MIN	Output without additional information.

Examples

1. Output of the contents starting from virtual address V’8200’ up to V’8203’ in dump format (if the storage type specification is omitted, %XL4 is used).

```
/%DISPLAY V'8200'
*** TID: 001901D2 *** TSN: 1544 ********************************************
CURRENT PC: 0000000C    CSECT: M1BS  ****************************************
V'00008200' = UPRONUM  + #‘000001E8'
00008200 (000001E8) 002C00C1                                ...A
```

2. Output of the contents starting with virtual address V’8200’ up to V’8203’ in character format (%C) and as a packed integer (%P) with a length of two bytes.

```
/%DISPLAY V'8200'%C
V'00008200' = UPRONUM  + #‘000001E8'
00008200 (000001E8) ...A

/%DISPLAY V'8200'%PL2
V'00008200' = UPRONUM  + #‘000001E8'
00008200 (000001E8)     +2
```

3. The instruction LA is located at address V’100’. In the first %DISPLAY, the address is modified only with storage type %SX, which causes 4 bytes as of address V’100’ to be output in hexadecimal form. If the address is to be calculated by AIDin the same way as the LA instruction would be executed by the hardware, then the address must be followed by the pointer operator (->) when entered in the %DISPLAY. The address is calculated as follows: the value of base register %3 and the displacement X’01B’ are added to the address in index register %1. The result is address V’725’, and the memory location thus addressed is output with a length of 4 bytes in dump format.

```
/%DISPLAY V'100'%SX
V'0000100' = M1BS  + #‘00000100'
00000100 (00000100) 4101301B

/%DISPLAY %1, %3
%1              = 00000700
%3              = 0000000A

/%DISPLAY V'100'%SX->
V'00000725' = M1BS     + #‘00000725'
00000725 (00000725) 58F0F008                                .00.
```

4. With keyword %PCB, AID accesses the process control block and outputs it in tabular form. With keyword %LOC, AID outputs information on static program nesting of the specified address.

```
/%DISPLAY %PCB
 P1 LEVEL STACK AT LOCATION 720EAAB0       PRESENT LOCATION: ABSOLUT +00000060
                                                (LAST CALLED SVC: 27=SYSF-MGT)
  LNK 73AC86D0   CWRD 86010200   (PC) 00000060    (0) 00000000    (1) 9F00003C
  (2) 00000002    (3) 00000000    (4) 00000000    (5) 00000003    (6) 00000000
  (7) 00000000    (8) 00000000    (9) 00000000    (A) 00000000    (B) 00000000
  (C) 00000000    (D) 00000000    (E) 00000000    (F) 00000000   FLP0 00000000
 FLP0 00000000   FLP2 00000000   FLP2 00000000   FLP4 00000000   FLP4 00000000
 FLP6 00000000   FLP6 00000000    CLK 00000000   CCLK 00000000   MIX0 00000000
 POST 00000000   MIX1 81000000   PCTR 00000000   AIDA 70F16000   MIX2 80500000
 EXRT 00000000   AUDM 00000000   MIX3 00000000   TLMW 00000000        --------
```

```
/%D %LOC(C=M1BS)
V'00000000' = CONTEXT:LOCAL#DEFAULT
              LMOD :  %ROOT
              SMOD :  M1BS
              OMOD :  M1BS
              CSECT : M1BS     (00000000) + 00000000
```

5. The first 3 examples illustrate the use of a simple memory reference, a one-level indirect address, and then a two-level indirect address. Finally, the two-level indirect addressing is followed by the byte offset with which the same address is calculated here as in the first %DISPLAY command. More information on the multiple use of a byte offset and indirect addressing can be found in section 7.2.4 of the AID Core Manual [1]].

```
/%D C=M1BS.#'A0'
V'000000A0' = M1BS      + #'000000A0'
000000A0 (000000A0) 000007FD                                ...}

/%D C=M1BS.#'A0'->
V'000007FD' = M1BS      + #'000007FD'
000007FD (000007FD) 00000000                                ....

/%D C=M1BS.#'A0'->->
V'00000000' = M1BS      + #'00000000'
00000000 (00000000) 58F0F008                                .00.

/%D C=M1BS.# 'A0'->->.#'A0'
V'000000A0' = M1BS      + #'000000A0'
000000A0 (000000A0) 000007FD                                ...}
```

6. The %DUMPFILE command is used to assign the dump file M.DUMP to the AID link name D1. The first 20 bytes, starting with address V'798' are to be output in dump format. Since the current base qualification E=VM applies, the base qualification E=D1 must be specified explicitly.

```
/%DUMPFILE D1=M.DUMP
/%D E=D1.V'798'%L20
** D1: M.DUMP ********************************************************************
V'00000798' = MOBS     + #'00000798'
00000798 (00000798) 003DF000 00000000 C1C2C3C4 C5C6C7C8    ..0.....ABCDEFGH
000007A8 (000007A8) C9D1D2D3                               IJKL
```

7. The following is to be output to SYSLST:
   - a header line with the current status of the program counter and the name of the CSECT in which the program has been interrupted (**%D P=MAX**).
   - the program counter (**%PC**)
   - a blank line (**%NL(1)**)
   - four bytes in dump format starting with address V'798'
   - a blank line (**%NL**; default value = 1)
   - all general-purpose registers (**%MR**)

   This is followed by positioning to the next page (**%NP**).

```
/%OUT %D P=MAX
/%D %PC, %NL(1), V'798', %NL, %MR, %NP
```

8. The last four bytes of class 6 memory were to be output by AID. Since these memory locations have not been allocated, AID outputs a corresponding error message.

```
/%DISPLAY %CLASS6.(%L(%CLASS6)-4)
V'003F8FFC' = %CLASS6  + #'003F8FFC'
AID0295 PAGE(S) FROM ADDRESS 003F8000 TO 003F8FFF NOT ALLOCATED
```

9. AID calculates the value as follows:
   To the last two bytes in register %1 add the value resulting from AID register %6G multiplied by 2, and to this add the length of CSECT CS1.
   Length modification %FL2 is required, otherwise the byte offset (.) would exceed the area limits of register %1.

```
/%D %L=(%1.2%FL2 + (%6G * 2) + %L(C=CS1))
```

10. ESA support
    The loaded program (E=VM) is working in AR mode, which means that addressing in data areas occurs via access registers.
    In the dump file ESA.AR.DUMP, by contrast, AR mode was not used when creating the dump, so all operations were only in the program area.

```
%DISPLAY %ASC
%ASC            = 01


%DF D3=ESA.AR.DUMP
%BASE E=D3
%DISPLAY %ASC
** D3: ESA.AR.DUMP ************************************************************
%ASC            = 00
```

11. ESA support
    In the first %DISPLAY, the memory location with address V’34’ in the program area is output in the type of the data definition C#DS11.
    All other %DISPLAY commands show the memory location with address V’34’ in the data area with SPID X’0000000080000200’ in the type of the data definition C#DS11. The ALET X’0001001B’ points to the same data area. It is buffered in %2G and held in %7AR. Consequently, all four %DISPLAY commands have the same effect.

```
/%DISPLAY V'34' %T(C#DS11)
SRC_REF:   288 SOURCE: DS2S4A    PROC: DS2S4A  ********************************
C#DS11          = |PS1-|

/%DISPLAY ALET=%2G.V'34' %T(C#DS11)
C#DS11          = |DS11|

/%DISPLAY ALET=X'0001001B'.V'34' %T(C#DS11)
C#DS11          = |DS11|

/%DISPLAY ALET=%7AR.V'34' %T(C#DS11)
C#DS11          = |DS11|

/%DISPLAY SPID=X'0000000080000200'.V'34' %T(C#DS11)
C#DS11          = |DS11|
```

12. ESA support
    A datta area is addressed via the ALET in access register 4. All virtual addresses (V’200’, V’0’, V’4’) from this data area are obtained in the following complex memory reference. AID calculates V’1B30’ + #’08’ = address V’1B38’ and outputs 17 bytes (X’11’) in dump format starting with that address.

```
%DISPLAY %AR, ALET=%4AR.V'200', ALET=%4AR.V'0', ALET=%4AR.V'10'
%AR   ( 0: 15)
(  0) 00000000  (  1) 00010003  (  2) 00010004  (  3) 00010005  (  4) 00010006
(  5) 00000000  (  6) 00000000  (  7) 00000000  (  8) 00000000  (  9) 00000000
( 10) 00000000  ( 11) 00000000  ( 12) 00000000  ( 13) 00000000  ( 14) 00000000
( 15) 00000000

V'00000200' = *ABSOLUT + #‘00000200'
00000200 (00000000) 00001B30                                    ....

V'00000000' = *ABSOLUT + #‘00000000'
00000000 (00000000) 0800C301                                    ..C.

V'00000010' = *ABSOLUT + #‘00000010'
00000010 (00000010) 110300D1                                    ...J


%D ALET=%4AR.V'200'->.(V'0' %FL1) %L(V'10' %XL1)
V'00001B38' = *ABSOLUT + #00001B38''
00001B38 (000004D0) D4E4C5D3 D3C5D940 C1D5D5C5 D3C9C5E2    MUELLER ANNELIES
00001B48 (000004E0) C5                                     E
```

13. ESA support
    A %DISPLAY with %DS is used to output the SPID, all the ALETs, and the size in bytes (SIZE) for all active data areas.
    When you specify an ALET or SPID qualification after the keyword, AID outputs the associated SPID or ALETs.

```
/%DISPLAY %DS
*** TID: 00020243 *** TSN: 13KG ********************************************
CURRENT PC: 000000E0      CSECT: DS2S4A     *********************************

SPID                ALET       SIZE

0001003300000031   00000000   0
0000000080000800   0001001B   409600
0000000080000800   0001001C   409600
0000000080000900   00010011   131072
0000000080000900   00010012   131072

/%DISPLAY %DS(SPID=X'0000000080000800')

     SPID  =  0000000080000800

     ALET  =  0001001B
     ALET  =  0001001C

/%DISPLAY %DS(ALET=X'00010011')

     ALET  =  00010011

     SPID  =  0000000080000900
```

%DUMPFILE

With %DUMPFILE you assign a dump file to one of the link names and cause AID to open or close this file.

– With *link* you select the link name for the dump file to be opened or closed.

– With *file* you designate the dump file to be opened.

Command	Operand
{%DUMPFILE / %DF}	[link [=file]]

If you omit the *file* operand AID will close the file assigned to the specified link name.

With a %DUMPFILE command without operands, you cause AID to close all open dump files. If the AID work area was contained in a dump file now closed, the AID standard work area reapplies (see %BASE command).

%DUMPFILE may only be specified as an individual command, i.e. it may not be part of a command sequence or subcommand.

The %DUMPFILE command does not alter the program state.

link

Designates one of the AID link names for dump files and has the format Dn, where n is a number with a value $0 \leq n \leq 7$.

file

Specifies the fully-qualified file name under which the dump file AID is to open is cataloged. If this operand is omitted, the dump file with the link name *link* is closed.
An open dump file must first be closed with a separate %DUMPFILE command before another file can be assigned to the same link name.

Examples

1. `%DUMPFILE D3=DUMP.1234.00001`
   The file with the BS2000 catalog name DUMP.1234.00001 is assigned to link name D3 and opened.
2. `%DF D3`
   The file assigned to link name D3 is closed.
3. `%DF`
   All open dump files are closed.

%FIND

With %FIND you can search for a literal in the user area of the loaded program or in a memory dump in a dump file and output hits to the terminal (via SYSOUT). In addition, the address of the hit and the continuation address are stored in AID registers %0G and %1G.

– *search-criterion* is the character literal or hexadecimal literal to be located.

– With *find-area* you specify which memory area AID is to search for *search-criterion*.

– With *alignment* you specify whether the search for *search-criterion* is to be effected at a doubleword, word, halfword or byte boundary. When a value for *alignment* is not given, searching takes place at the byte boundary.

– With *ALL* you specify that the search is not to be terminated after output of the first hit, rather the entire *find-area* is to be searched and all hits are to be output. The search can only be aborted by pressing the K2 key.

Command	Operands
%F[IND]	[[ALL] search-criterion [IN find-area] [alignment]]

If the *ALL* operand is omitted from a %FIND command, the user may continue after the address of the last hit and up to the end of the *find-area* by specifying a new %FIND command without any operand values.

If a %FIND command is issued with a separate *search-criterion* and without any further operands, AID inserts the appropriate default value for operands that have no current value, i.e. does not transfer operands from a preceding %FIND command in this case.

The output of hits is always in hexadecimal character representation, with a maximum length of 12 bytes, andd is sent to the terminal (SYSOUT). In addition to the hit itself, its address and (insofar as possible) the name of the CSECT in which the hit was found, and the CSECT-relative address of the hit are output.

In the event of a hit, the hit address is stored in AID register %0G and the continuation address (hit address + search string length) in AID register %1G. With the *ALL* specification, the address of the last hit is stored in %0G and the continuation address of the last hit is stored in %1G. If the *search-criterion* has not been found, AID register %1G remains unchanged. AID register %0G is set to -1.
The two register contents permit you to use the %FIND command in procedures as well as in subcommands and to further process the results.

The %FIND command does not alter the program state.

search-criterion

Character literal or hexadecimal literal which can be searched for in virtual memory or in a dump file. *search-criterion* may contain wildcard symbols. These symbols are always hits. They are represented by '%'.

```
search-criterion-OPERAND  - - - - - - - - - - - - - - - - - - - - - - - - - - - -

{C'x...x' | 'x...x'C | 'x...x'}
{X'f...f' | 'f...f'X         }

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

`{C'x...x' | 'x...x'C | 'x...x'}`
: Character literal with a maximum length of 80 characters. Lowercase letters can only be searched for by deactivating conversion to uppercase, i.e. specifying %AID LOW[=ON].

  x can be any representable character, in particular the wildcard symbol '%', which always represents a hit. The character '%' itself cannot be located when it is in this form, since C'%' in a character literal must always result in a hit. For this reason it must be represented as the hexadecimal literal X'6C'.

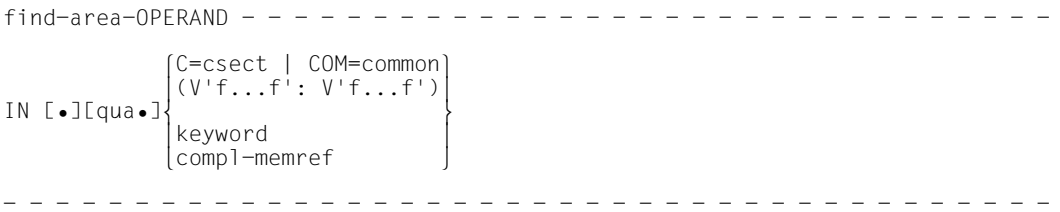
`{X'f...f' | 'f...f'X}`
: Hexadecimal literal with a maximum length of 80 digits or 40 characters. A literal with an odd number of digits is padded with X'0' on the right.

  f can assume any value between 0 and F, as well as the wildcard symbol X'%'. The wildcard symbol represents a hit for any hexadecimal digit between 0 and F.

find-area

Defines the memory area to be searched for *search-criterion*. *find-area* can be either a subarea of virtual memory or a subarea of a dump file.

If no *find-area* is specified, AID inserts the default value %CLASS6 and searches the entire class 6 memory in the current AID work area; in the case of XS programs, it searches only the lower address space.



- If the period is in the leading position it denotes a *prequalification*, which must have been defined with a preceding %QUALIFY command. Consecutive qualifications must be separated by a period. In addition, there must be a period between the final *qualification* and the following operand part.

qua One or more qualifications may be specified here if a memory object to be addressed is not located in the current AID work area or is not unique in that area. It is sufficient to specify only the qualifications needed for a unique address.

E={VM | Dn}
: Specified only if the current base qualification does not apply to *find-area* (see %BASE).

{ALET={X'f...f' | %nAR | %nG} | SPID=X'f...f'}
: Specified to reference an address in a data area, but only required when debugging ESA programs. These qualifications must be followed by a V address or a *compl-memref*.

CTX=context
: Specified only to enable unique addressing of a CSECT or a COMMON in the program system.

[L=load-unit•][O=object-module•]
: Specified only if a CSECT or COMMON to be referenced cannot be uniquely addressed in the current context without these qualifications. You only need to specify the qualifications required for a unique address.

{C=csect | COM=common}
: If addressing ends with a C/COM qualification, the specified CSECT or COMMON is defined as the *find-area*.

keyword

Defines a memory area by one of the following keywords (see chapter 10 in the AID Core Manua [1]). In the case of XSand ESA systems, %CLASS6 and %CLASS6BELOW designate the same address area.

keyword	Meaning
%CLASS6	Class 6 memory
%CLASS6BELOW	Class 6 memory below the 16-Mb limit
%CLASS6ABOVE	Class 6 memory above the 16-Mb limit

(V'f...f':V'f...f')

Defines a memory area by means of a virtual start address and a virtual end address. The resulting address difference must not exceed a value of 65535 and the start address must be less than or equal to the end address.
f...f is a hexadecimal address with up to eight digits.

compl-memref

Designates a 4-byte area starting at the calculated address. If a different number of bytes are to be searched, *compl-memref* must end with the appropriate length modification (≤ **64 KB**). The following operations may appear in *compl-memref* (see chapter 7 in the AID Core Manual [1] and chapter 4, "Machine-code-specific addressing" in this manual for restrictions on debugging ESA programs):

- byte offset (•)
- indirect addressing (->)
- type modification (%A, %S, %SX)
- length modification (%L(...), %L=(expression), %Ln)
- address selection (%@(...))

alignment

Defines that *search-criterion* is only to be searched for at certain aligned addresses.

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

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

search-criterion is searched for:

1 at byte boundary (default)

2 at halfword boundary
4 at word boundary
8 at doubleword boundary

Examples

1. `%FIND C'***' IN %CLASS6`
   The character literal C'***' is searched for in class 6 memory (below the 16-Mb boundary) of the currently valid base qualification. Any hit is output to SYSOUT.

2. `%F`
   The search is continued with the parameters of the last %FIND command after the last hit.

3. `%F ALL X'00F%00' ALIGN 2`
   At each halfword boundary a hexadecimal literal which has X'00F' in the first three halfbytes and an arbitrary hexadecimal value (represented by %) in the fourth halfbyte and ends with X'00' is searched for.

4. `%ID LOW = ON`
   `%IN V'8A0' <(%0G NE -1):%MOVE 'Donna' INTO %0G->>`
   `%IN V'8A0' <%FIND 'Billy'>`
   Differentiation between uppercase and lowercase is activated.
   By chaining, the following command sequence is created at test point V'8A0':
   `%FIND 'Billy'; (%0G NE -1):%MOVE 'Donna' INTO %0G->`
   The condition may only be written at the beginning of a subcommand. For this reason two %INSERT commands are required for the same test point if only part of the command sequence to be effective at the test point is to be dependent on the condition. The string 'Billy' is to be searched for at the test point with virtual address V'8A0'. In the event of a hit (AID register %0G ≠ -1) the corresponding memory area is overwritten with 'Donna'.

5. `%IN V'1648' <(%0G NE -1): %SET %L=(%1G-%0G) INTO %2G>`
   `%IN V'1648' <%FIND 'x...x'>`
   This command sequence can be used in a procedure in order to search for any arbitrary character literal. In the event of a hit, the address which has been found is stored in AID register %0G and the length of the located string in %2G.

6. `%F X'ABCD' IN V'A1A'%L=(%1)`
   An arbitrary character literal is searched for in a memory area of variable length, starting with address V'A1A'. The length of the relevant memory area is stored in register 1.

%HELP

By means of %HELP you can request information on the operation of AID. The following information is output to the selected medium: either all the AID commands or the selected command and its operands, or the selected error message with its meaning and possible responses.

– By means of the *info-target* operand you specify the command on which you need further information or the AID message for which you want an explanatory text with a description of actions to be taken.

– By means of the *medium-a-quantity* operand you specify to which output media AID is to output the required information. By means of this operand you overwrite any declaration made via %OUT, but only for the current %HELP command.

Command	Operand	
%H[ELP]	[info-target]	[medium-a-quantity][,...]

%HELP provides information on all operands of the selected command, i.e. on both the language-specific operands for symbolic debugging and the operands for machine-oriented debugging. Refer to the relevant language-specific manual to see what is applicable for the particular language in which your program is written.

The format for the message key for AID messages is AID0*n*; the format for AIDSYS messages is IDA0*n*. Both can be requested with /HELP-MESSAGE-INFORMATION. Furthermore, the AID command %HELP can be used in this version to reference messages with the old message key *In*.

Messages from AIDSYS have the message code format IDA0nnn and are queried using the /HELP command.

%HELP can only be entered as an individual command, i.e. it must not be contained in a command sequence or subcommand.

The %HELP command does not alter the program state.

info-target

Designates a command or a message number on which information is to be output.
If the *info-target* operand is omitted, the command initiates output of an overview of the AID commands with a brief description of each command, and of the AID message number range.

AID responds to a %HELP command having an invalid *info-target* operand by issuing an error message. This is followed by the same overview as for a %HELP command without *info-target*. This overview can also be requested via the %?, %H? or %H %? entries.

```
info-target-OPERAND - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

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

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

The AID command names may be abbreviated as shown above.

In designates the old number for which the meaning and possible actions are to be output.
n is the three-digit message number. Note that the use of the AID %HELP command will not be supported in future AID versions, since the same information can be requested with the SDF command `/HELP-MESSAGE-INFORMATION`.

medium-a-quantity

Defines the media via which information on the *info-target* is to be output.

If this operand is omitted and no declaration has been made using the %OUT command, AID works with the default value T=MAX.

```
medium-a-quantity-OPERAND - - - - - - - - - - - - - - - - - - - - - - - - - - -

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

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

medium-a-quantity is described in detail in chapter 8 of the AID Core Manual [1].

T Terminal output
H Hardcopy output (includes terminal output; cannot be specified in conjunction with *T*)

Fn File output
P Output to SYSLST

The {MAX | MIN} specification has no effect for %HELP, but one of the two entries is required for syntactical reasons.

%INSERT

By means of %INSERT you can specify a test point and define a subcommand. Once the program sequence reaches the test point, AID processes the associated subcommand. In addition, the user can also specify whether AID is to delete the test point once a specific number of executions has been counted and whether or not to interrupt the program afterwards.

– By means of the *test-point* operand you may define a program instruction prior to whose execution AID interrupts the program run and processes *subcmd*.

– By means of the *subcmd* operand you may define a command or a command sequence and perhaps a condition. Once *test-point* has been reached and the condition satisfied, *subcmd* is processed.

– By means of the *control* operand, you can declare whether *test-point* is to be deleted after a specified number of passes and whether the program is then to be halted.

Command	Operand		
%IN[SERT]	test-point	[<subcmd>]	[control]

A *test-point* is deleted in the following cases:

1. When the end of the program is reached.
2. When the number of passes specified via *control* has been reached and deletion of *test-point* has been specified.
3. If a %REMOVE command deleting the *test-point* has been issued.

If no *subcmd* operand is specified, AID inserts the *subcmd* <%STOP>.

The *subcmd* in an %INSERT command for a *test-point* which has already been set does not supersede the existing *subcmd*. This means that subcommands for the same *test-point* are processed according to the LIFO rule (last in, first out).

The %REMOVE command is used to delete a subcommand, a *test-point* or all *test-point* operands which have been entered. *test-point* can only be an address in the program which has been loaded, for which reason the base qualification E=VM must be valid (see %BASE) or specified explicitly.

The %INSERT command does not alter the program state.

test-point

Is the address of an executable machine instruction. *test-point* is immediately entered by targeted overwriting of the memory location addressed and must therefore be present in virtual memory at the time the %INSERT command is input. A *test-point* which is not entered for the start address of an executable machine instruction will lead to errors in the program run (e.g. data/addressing errors).

When the program reaches the *test-point*, AID interrupts the program run and starts the *subcmd*. If the program is continued it begins with execution of the instruction overwritten by *test-point*.

```
test-point-OPERAND  - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

                   ⎧ C=csect | COM=common⎫
[•][qua•]          ⎨ V'f...f'            ⎬
                   ⎩ compl-speicherref   ⎭

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

- If the period is in the leading position it denotes a *prequalification*, which must have been defined with a preceding %QUALIFY command.
  Consecutive qualifications must be separated by a period. In addition, there must be a period between the final *qualification* and the following operand part.

qua One or more qualifications may be specified here if a memory object to be addressed is not located in the current AID work area or is not unique in that area. It is sufficient to specify only the qualifications needed for a unique address.

E=VM
: Since *test-point* can only be entered in the virtual memory of the program which has been loaded, specify *E=VM* only if a dump file has been declared as the current base qualification (see %BASE).

CTX=context
: Specified only to enable unique addressing of a CSECT or a COMMON in the program system.

[L=load-unit•][O=object-module•]
: Specified only if a CSECT or COMMON to be referenced cannot be uniquely addressed in the current context without these qualifications. You only need to specify the qualifications required for a unique address.

{C=csect | COM=common}

If addressing ends with a C/COM qualification, the *test-point* is placed at the start address of the designated CSECT or COMMON.

V'f...f'

Is a virtual address, where *f...f* is a valid hexadecimal address with up to eight digits. The address must be the start address of an interpretable machine instruction, otherwise errors (e.g. data or addressing errors) may occur.

compl-memref

The result of *compl-memref* must be the start address of an executable machine instruction. The following operations may occur in *compl-memref* (see section 7.2.4 in the AID Core Manual [1]):

- byte offset (•)
- indirect addressing (->)
- type modification (%A, %S, %SX)
- length modification (%L(...), %L=(expression), %Ln)
- address selection (%@(...))

subcmd

subcmd is processed whenever the program run reaches the address designated by *test-point*.
If the *subcmd* operand is omitted, AID inserts a %STOP command.

```
subcmd-OPERAND  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

                                      ⎧AID-command    ⎫
< [subcmdname:] [(condition):] [⎨                     ⎬ {;...}] >
                                      ⎩BS2000-command ⎭

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

The subcommand can contain a name, a condition and a command portion. The command part may consist of a single command or a command sequence, it can contain both AID and BS2000 commands as well as comments. There is an execution counter for each subcommand. Refer to the AID Core Manual, chapter 5, for more information on how an execution condition is formulated, how the name and execution counter are assigned and addressed, and which particular commands are not permitted within subcommands.

If a subcommand consists of only a name or a condition, i.e. if the command part is missing, AID merely increments the execution counter when *test-point* is reached.

subcmd does not overwrite an existing subcommand for the same *test-point*, but the new subcommand is prefixed to the existing one. *subcmd* may contain the commands %CONTROLn, %INSERT and %ON. Nesting over a maximum of 5 levels is possible. Subcommand chaining and nesting is described in more detail in the AID Core Manual, chapter 5.

The commands in a *subcmd* are executed one after the other and the program run is then continued. The commands for runtime control immediately alter the program state, even in a subcommand. They abort the *subcmd* and continue the program (%CONTINUE, %RESUME, %TRACE) or halt it (%STOP). They are thus only effective as the last command in a *subcmd*, since any subsequent commands in the *subcmd* would fail to be be executed.

control

Specifies whether *test-point* is to be deleted after the n-th pass, and whether the program is to be halted with the purpose of entering new commands.
If no *control* operand has been specified, AID assumes the defaults 2^{31}-1 (for *n*) and *K*.

Note that this operand will not be described in later editions.

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

                  ⎧ K ⎫
ONLY  n  [⎨ S ⎬]
                  ⎩ C ⎭

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

n Is an integer with the value $1 \leq n \leq 65535$, specifying after how many *test-point* passes the further declarations for this *control* operand are to go into effect. The default is 65535.

K Neither *subcmd* nor *test-point* is deleted (KEEP).
Program execution is interrupted, and AID expects input of commands.

S *subcmd* is deleted. If no further subcommand for this *test-point* has been entered, *test-point* is deleted as well.
Program execution is interrupted, and AID expects input of commands (STOP).

C *subcmd* is deleted. If no further subcommand for this *test-point* has been entered, *test-point* is deleted as well.
No interruption of the program (CONTINUE).

Examples

1. `%IN V'80' <SUB1: %DISPLAY %5>`
   `%IN V'80' <SUB2: %DISPLAY %7> ONLY 4 S`
   When the address V’80’ is reached, the contents of registers 7 and 5 are output. If the subcommand SUB2 has been executed four times, it is deleted and the program is interrupted. SUB1 is retained and is executed during the course of further debugging whenever address V’80’ is reached.

2. `%IN C=CS1.16 <(%0G NE -1): %SET %L=(%1G-%0G) INTO %2G>`
   A test point is entered for the 16th byte of CSECT CS1. If program execution reaches this address, the return messages of the last %FIND command executed are checked. If no hit was found, AID register %0G is set to -1, the condition result is FALSE and the subcommand is not executed; if a hit was found, the address of the hit is stored in AID register %0G, the condition result is TRUE and the subcommand is executed. The length of the search string is calculated with the aid of the length function and transferred to AID register %2G.

3. `%IN V'800'%SX-> <%DA 2 FROM %PC->>`
   Address V’800’ contains an Assembler instruction (RX format). A test point is entered for the address resulting from the index register, base register and displacement. If the program run reaches this address, the next two instructions are disassembled.

%MOVE

With the %MOVE command you transfer memory contents or AID literals to memory locations within the program which has been loaded. Transfer is effected bytewise, left-justified, without checking and matching of sender and receiver storage types.

– With the *sender* operand you designate the memory location whose contents are to be transferred, a length, an address or an AID literal. *sender* can be located in virtual memory of the loaded program or in a dump file.

– With the *receiver* operand you designate the memory location to be overwritten. *receiver* can only be located in virtual memory of the loaded program.

– With the *REP* operand you specify whether AID is to generate a REP record for a modification effected. This operand supersedes any declaration made in the %AID command, but only for the current %MOVE command.

Command	Operand
%M[OVE]	sender INTO receiver [REP]

In contrast to the %SET command, AID does not check for compatibility between the storage types of *sender* and *receiver* when the %MOVE command is involved, and does not match the type of *sender* to that of *receiver*.

AID transfers the information left-justified, with the length of *sender*. If the length of *sender* is greater than that of *receiver*, AID rejects the attempt to transfer and issues an error message.

Using the %AID command (*CHECK* operand) you can activate an update dialog for checking purposes, which first provides you with a display of the old and new contents of *receiver* and offers you the option of aborting the %MOVE command.

The %MOVE command does not alter the program state.

sender INTO receiver

For *sender* and *receiver* you can specify a virtual address, a C/COM qualification, a complex memory reference or a keyword. AID literals, selectors, and the keyword %MODEn can only be specified as *sender*. Note that moving or overwriting instruction code may have undesirable side-effects if *sender* or *receiver* lies in a *control-area* or *trace-area* that was declared symbolically (see section 12.2 in the AID Core Manual [1]).

sender may be either in virtual memory or in a dump file; *receiver* can only be in virtual memory.

No more than 3900 bytes can be transferred with a %MOVE command. If the area to be transferred is larger, multiple %MOVE commands must be issued.

```
sender-OPERAND  - - - - - - - - - - - - - - - receiver-OPERAND  - - - - - - - - -

 { [          {C=csect | COM=common}   }                                        }
 { [          {keyword                 }                                        }
 { [•][qua•]{                          } }                                        }
 { [          {V'f...f'                } }                                        }
 { [          {compl-memref            } }                                        }
 {                                       } INTO [•][qua•]{C=csect | COM=common}   }
{  %@(...)                               }               {keyword                 }
{  %L(...)                               }               {                        } }
{  %L=(expression)                       }               {V'f...f'                } }
 {                                       }               {compl-memref            } }
 {  AID-literal                          }                                        }
 {                                       }                                        }
 {  %M[ODE]{24|31}                         INTO           %AMODE                   }
```

- If the period is in a leading position it identifies a *prequalification*, which must have been defined by a preceding %QUALIFY command. A period must be inserted between consecutive qualifications. In addition, a period must be located between the last *qualification* and the subsequent operand part.

qua One or more qualifications may be specified here if a memory object to be addressed is not located in the current AID work area or is not unique in that area. It is sufficient to specify only the qualifications needed for a unique address.

{E={VM | Dn} for *sender* | E=VM for *receiver*}
A base qualification need only be specified if the current base qualification is not to apply. *sender* can be either in virtual memory or in a dump file. *receiver*, on the other hand, can only be located in virtual memory.

{ALET={X'f...f' | %nAR | %nG} | SPID=X'f...f'}
Specified to reference an address in a data area, but only required when debugging ESA programs. These qualifications may only be followed by a V address or a *compl-memref*.

CTX=context
Specified only to enable unique addressing of a CSECT or a COMMON in the program system.

[L=load-unit•][O=object-module•]
Specified only if a CSECT or COMMON to be referenced cannot be uniquely addressed in the current context without these qualifications. You only need to specify the qualifications required for a unique address.

{C=csect | COM=common}
If addressing ends with a C/COM qualification, the entire selected program segment is transferred (*sender*), or overwritten from the beginning (*receiver*). However, *sender* must not be longer than 3900 bytes.

V'f...f'
Specifies a virtual address.

compl-memref
The following operations may occur in *compl-memref* (see section 7.2.4 in the AID Core Manual [1] and chapter 4 in this manual for restrictions on debugging ESA programs):

– byte offset (•)
– indirect addressing (->)
– type modification (%T(dataname), %X, %C, %P, %D, %F, %A, %S, %SX)
– length modification (%L(...), %L=(expression), %Ln)
– address selection (%@(...))

A subsequent type modification has no effect on the transfer, as the %MOVE command always transfers data in binary form, regardless of the *sender* and *receiver* storage type.

A following length modification for *sender* defines how many bytes are to be moved. A length modification for *receiver* can become necessary, however, in order to avoid exceeding the area limits of *receiver*.

Data names and statement names defined in the source program can be used within *compl-memref* if LSD records exist.

keyword

Designates a register, the program counter or an execution counter. See chapter 10 in the AID Core Manual [1] for a description of the implicit storage types of keywords.

keyword	Meaning
%PC	Program counter
%n	General register, $0 \leq n \leq 15$
%nD|E	Floating-point register, single/double precision, n = 0,2,4,6
%nQ	Floating-point register, quadruple precision, n = 0,4
%nG	AID general register, $0 \leq n \leq 15$
%nDG	AID floating-point register, n = 0,2,4,6
%•[subcmdname]	Execution counter. The abbreviation %. designates the execution counter of the currently active subcommand.

In addition, the keywords for storage classes may be used (see section 10.5 in the AID Core Manual [1]).

%@(...)

With the address selector you can use the storage class (%CLASS6; %CLASS5 for class 5 memory) or *compl-memref* as *sender*.
The address selector supplies an address constant which you can use in a complex memory reference before a pointer operator (->). See also section 7.2.4.4 in the AID Core Manual [1].

%L(...)

With the length selector you can use the length of a CSECT, a COMMON, or a storage class (%CLASS6, %CLASS6BELOW, %CLASS6ABOVE) as *sender*.
The length selector returns an integer that may be used for byte offset calculations or within a length function (see section 7.2.4.4 in the AID Core Manual 1]).
Example: With %L(C=CS1) the length of CSECT CS1 is used as *sender*.

%L=(expression)

With the length function you can calculate a value as *sender*. *expression* is comprised of the contents of memory references, constants, integers and arithmetic operators. Only memory reference contents of type %F or %A with a length less than or equal to 4 are permitted. The length function produces an integer, which can be used for byte offset calculation, in a further length function or for length modification (see section 7.2.4.4 in the AID Core Manual [1]).
Example: With %L=(%1) the contents of register 1 are used as *sender*.

AID-literal

The following AID literals (see chapter 9 of the AID Core Manual [1]) can be transferred with %MOVE:

```
{C'x...x' | 'x...x'C | 'x...x'}   Character literal
{X'f...f' | 'f...f'X}             Hexadecimal literal
{B'b...b' | 'b...b'B}             Binary literal
[{±}]n                            Integer
#'f...f'                          Hexadecimal number
[{±}]n.m                          Decimal point number
[{±}]mantissaE[{±}]exponent       Floating-point number
```

%M[ODE]{24|31}

Defines the addressing mode.

%AMODE

Is the system information field in which the addressing mode is entered. Transfer of keyword %M{24|31} to the system information field %AMODE changes the addressing mode of the test object. If only the address interpretation for indirect addressing (->) is to be modified, the corresponding %AINT command should be entered.

If the program counter is to be modified when testing a 31-bit-address program while 24-bit mode is switched on, care must be taken to ensure that the most significant byte has the value X'00'.

REP

Specifies whether AID is to generate a REP record after a modification has been performed. With *REP* you temporarily deactivate a declaration made with the %AID command. If *REP* is not specified and there is no valid declaration in the %AID command, no REP record is created.

```
REP-OPERAND - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

REP = {Y[ES] | N[O]}

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

REP=Y[ES]

LMS correction statements (REPs) are created in SDF format for the update caused by the current %MOVE command. A MODIFY-ELEMENT statement must be inserted for the LMS run. If the object structure list is not available, no correction statements are generated, and AID will output an error message. If *receiver* is not located completely within one CSECT, AID will also output an error message and

not write a REP record. To obtain REP records in the latter case, the user should distribute transfer operations over several %MOVE commands in which the CSECT limits are observed.
A REP record is stored in a file with link name F6. If no file with the link name F6 has been created (see %OUTFILE command), the REP record is stored in the file AID.OUTFILE.F6 generated by AID.
If you create REP records, you should reserve link name F6 only for the %MOVE command, in order not to inadvertently write other test data to the REP file.

REP=N[O]
No REP records are generated for the current %MOVE command.

Examples

1. `%MOVE %6 ->.20 INTO V'10A'`
   The start address of *sender* is calculated from the contents in general register 6, to which 20 is added. This command is used to transfer four bytes (standard length of a memory location).

2. `%MOVE E=D1.%PC INTO V'A04'`
   The program counter status is transferred from the memory dump in the dump file with link name D1 to virtual address V'A04' of the current AID work area.

3. `%MOVE %7.1%L2 INTO %3.2%L2`
   The contents of bytes 2 and 3 of general register 7 are transferred to bytes 3 and 4 of general register 3. The length modification for *receiver* is required because the area limits of register 3 would be exceeded as a result of the byte offset.

4. `%MOVE X'47F0' INTO V'20' REP=YES`
   The contents of address V'20' are overwritten with hexadecimal literal X'47F0'. For this modification a REP record is generated and stored in the file AID.OUTFILE.F6 or the file assigned to link name F6.

5. `%MOVE %MODE31 INTO %AMODE`
   The addressing mode is set to 31-bit addresses for the current test object. The address interpretation for indirect addresses in AID commands is automatically adapted to the new addressing mode, provided no %AINT command is active. This declaration may also need to be adapted accordingly.

%ON

With the %ON command you define events and specify subcommands. If a selected event occurs in the program sequence, AID processes the associated subcommand.

– With *write-event* you define an event involving write access to a memory area. This means that AID will interrupt program execution to process *subcmd* whenever the program modifies the specified memory area.

– With *event* you define normal and abnormal program termination, a supervisor call (SVC), a program error or any event for which AID is to interrupt the program in order to process *subcmd*.

– With *subcmd* you define a command or a command sequence and perhaps a condition. When *event* occurs and this condition is satisfied, *subcmd* is executed.

Command	Operand	
%ON	event	[<subcmd>]

If the *subcmd* operand is omitted, AID inserts the *subcmd* <%STOP>.

The *subcmd* of an %ON command for an *event* which has already been defined does not supersede the existing *subcmd*, but the new *subcmd* is prefixed to the existing one. This means that subcommands for the same *event* are processed in accordance with the LIFO principle (last in, first out).
This does not apply to *write-event*. Each new *write-event* overwrites the previous one.

A defined event remains in effect until it is deleted with %REMOVE or the program terminates.

For %ON the base qualification E=VM must be set (see %BASE).

The %ON command does not alter the program state.

write-event

Write monitoring is activated with the keyword %WRITE. The memory area to be monitored is defined within parentheses after the keyword. As soon as the program changes a byte within the defined area, program execution is interrupted, and *subcmd* is executed. The interrupt point is located **after** the instruction responsible for the write access.

Only one *write-event* can be defined at any given time; the input of a new *write-event* overwrites the existing one. Other events can, however, be active at the same time. If an *event* and *write-event* occur simultaneously, AID will process the subcommand for *write-event* first. The *write-event* can be deleted with %REMOVE %WRITE without specifying the memory reference.

The following **interactions** must be noted between %ON *write-event* and other AID commands:

– If a %CONTROL*n* or %TRACE with a **machine-oriented** *criterion* is active, the input of %ON *write-event* is rejected with an error message, and vice versa.
– If a %CONTROL*n* or %TRACE with a **symbolic** *criterion* overwrites a machine instruction with the AID-internal marker (X'0A81'), the write access of that instruction is **not** detected by AID.
– AID **does not** detect the write access of a machine instruction at whose address a test point was set with %INSERT.

To provide for continuous write monitoring, it is advisable to delete all %CONTROL*n* and %INSERT commands with %REMOVE. An existing %TRACE, if any, is deleted when you continue after the %ON with %RESUME or %TRACE 1 %INSTR.

The memory area to be monitored can be any memory object, regardless how it is addressed, provided it is located within the memory area occupied by the program. Registers, for example, cannot be monitored. The area to be monitored **must not exceed 64KB**; otherwise, an error message is issued.
If the address of the specified memory object is overloaded in a program with an overlay structure, the corresponding area of the newly loaded program segment will be monitored.

Only the memory objects in the program area can be monitored when degubbing ESA programs.

```
write-event-OPERAND  - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

                    ⎧C=csect | COM=common⎫
                    ⎪V'f...f'            ⎪
%WRITE([•][qua•]⎨                        ⎬)
                    ⎪keyword             ⎪
                    ⎩compl-memref        ⎭

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

- If the period is in a leading position it identifies a *prequalification*, which must have been defined by a preceding %QUALIFY command. A period must be inserted between consecutive qualifications. In addition, a period must be located between the last *qualification* and the subsequent operand part.

qua One or more qualifications may be specified here if a memory object to be addressed is not located in the current AID work area or is not unique in that area. It is sufficient to specify only the qualifications needed for a unique address.

E={VM | Dn}
: Specified only if the current base qualification does not apply to the memory area to be monitored.

CTX=context
: Specified only to enable unique addressing of a CSECT or a COMMON in the program system.

[L=load-unit•][O=object-module•]
: Specified only if a CSECT or COMMON to be referenced cannot be uniquely addressed in the current context without these qualifications. You only need to specify the L and/or O qualifications required for a unique address.

{C=csect | COM=common}
: If addressing ends with a C/COM qualification, the entire program segment that was selected is monitored.

V'f...f' Designates a virtual address. The first 4 bytes as of the specified address are monitored.

compl-memref
: Designates the area to be monitored as of a calculated address in the appropriate implicit or specified length. The following operations may occur in *compl-memref* (see chapter 7 in the AID Core Manual [1] and chapter 4, "Machine-code-specific addressing" in this manual for restrictions on debugging ESA programs):

  – byte offset (•)
  – indirect addressing (->)
  – type modification (%A, %S, %SX)
  – length modification (%L(...), %L=(expression), %Ln)
  – address selection (%@(...))

keyword
: Allows you to define a memory area by specifying one of the following keywords (see chapter 9 in the AID Core Manual [1]). On XS and ESA computers, %CLASS6 and %CLASS6BELOW define the same address area.

```
%CLASS6               class 6 memory
%CLASS6BELOW          class 6 memory below the 16MB boundary
%CLASS6ABOVE          class 6 memory above the 16MB boundary
```

event

A keyword is used to specify an event (normal or abnormal termination of the program, supervisor call, program error, etc.) upon which AID is to process the *subcmd* specified. Once an event has been processed by an STXIT routine, it is no longer possible to respond to it with a *subcmd* assigned to that *event.*
When a subcommand is executed for the event %ANY, the prompt to check whether a dump is to be output is dropped in the close handling of the program. If required, you will need to explicitly initiate the output of the dump by using `/CREATE-DUMP` in the subcommand.

If several %ON commands with different *event* declarations are simultaneously active and satisfied, AID processes the associated subcommands in the order in which the keywords are listed in the table below. If various %TERM events are applicable, the associated subcommands are processed in accordance with the LIFO principle.
For selection of the suitable event codes and SVC numbers, see the "Executive Macros" manual [7].

If a *write-event* occurs together with some other *event* , the subcommand for *write-event* is processed first.

event	*subcmd* is processed in the following instances:
%ERRFLG (zzz)	after the occurrence of an error with the event code zzz and before abortion of the program
%INSTCHK	after the occurrence of an addressing error, an impermissible supervisor call (SVC), an operation code which cannot be decoded, a paging error or a privileged operation and before abortion of the program
%ARTHCHK	after the occurrence of a data error, divide error, exponent overflow or a zero mantissa and before abortion of the program
%ABNORM	after the occurrence of one of the errors covered by the previously described events as well as a DMS error (as of BS2000 V10) and %ILLSTX
%ERRFLG	after the occurrence of an error with any event code
%SVC(z...z) %SVC	before execution of the supervisor call (SVC) with the specified number before execution of any supervisor call

event	*subcmd* is processed in the following instances:
%LPOV(x...x) %LPOV	after loading of the module or segment with the specified name after loading of any arbitrary module/segment (The name is output with %D %LINK)
%TERM(N[ORMAL]) %TERM(A[BNORMAL] %TERM(D[UMP]) %TERM(S[TEP]) %TERM	before normal program termination before abnormal program termination, but after output of a dump before output of a dump; followed by program termination before program termination; followed by a branch within procedures before program termination by all of the %TERM events described above
%ANY	before termination of the program as the result of a program error or as the result of the %TERM events described above, or due to a DMS error (as of BS2000 V10) or a %ILLSTX
%ILLSTX	before the occurrence of a STXIT call during processing of a preceding STXIT call (STXIT within STXIT)

zzz may be specified in one of two formats:
n unsigned decimal number (up to three digits)
#'ff' two-digit hexadecimal number
The following applies for the value *zzz*: $1 \leq zzz \leq 255$
No check is made to verify whether the specified event code or SVC number is meaningful or permissible.

x...x is the name of a segment or load unit (up to 32 alphanumeric characters).

subcmd

Is processed whenever the specified *event* occurs in the course of the program run. If the *subcmd* operand is omitted, AID inserts a %STOP command.
A complete description of *sbcmd* can be found in chapter 6 of the AID Core Manual [1].

```
subcmd-OPERAND  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

                                                 ⎧AID-command    ⎫
< [subcmdname:] [(condition):] [⎨               ⎬ {;...}] >
                                                 ⎩BS2000-command⎭

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

The subcommand can contain a name, a condition and a command portion. The command part can consist of either an individual command or a command sequence; it may contain AID and BS2000 commands as well as comments. Each subcommand has an execution

counter. Refer to the AID Core Manual, chapter 5, for information on how an execution condition is formulated, how the name and execution counter are assigned and addressed as well as which commands are not permissible within subcommands.

If a subcommand consists of only a name or a condition, AID merely increments the execution counter when the associated event occurs.

subcmd does not overwrite an existing subcommand for the same *event*. Instead, the new subcommand is prefixed to the existing one. The %CONTROLn, %INSERT and %ON commands are permitted in *subcmd*. The user can form up to five nesting levels. Subcommand chaining and nesting are described in the AID Core Manual, chapter 5.

The commands in a *subcmd* are executed one after the other; then the program is continued. The commands for runtime control immediately alter the program state, even in a subcommand. They abort *subcmd* and continue the program (%CONTINUE, %RESUME, %TRACE) or interrupt it (%STOP). They should only be issued as the last command in a *subcmd*, since any subsequent commands in that *subcmd* will not be executed.

Examples

1. `%ON %ERRFLG (108)`
   `%ON %ERRFLG (#'6C')`
   Both specifications designate the same program error (mantissa equal to zero).

2. `%ON %LPOV <%DISPLAY %LINK; %CONTINUE>`
   After each segment is loaded, the name of the loaded segment is output and the program is continued.

3. `%ON %ERRFLG <%D %AUD1;%STOP>`
   When an error occurs, regardless of the error weight, the P1 audit table is output and the program is halted (see the manual "User Commands (SDF Format)" [6]).

4. `%ON %WRITE(V'100') <%DA 4 FROM %PC–>.(–6); %DA 4 FROM %PC–>.(–4);%STOP>`
   Whenever one of the 4 bytes from V’100’ to V’103’ is modified in the program, the program is halted, and 4 disassembled instructions are output twice: starting with the address 6 bytes before the interrupt point, and starting with the address 4 bytes before the interrupt point. The interrupt point itself is located after the instruction that executed the write operation. The program is then halted, and a STOP message is issued.

5. `%ON %WRITE(V'200'%L2) <(V'200'%L2 EQ X'FFFF'):%STOP>`
   AID monitors byte V’200’ and V’201’. If the content of both bytes is X’FFFF’, the program is halted, and a STOP message is issued.

%OUT

With the %OUT command you define the media via which data is to be output and whether output is to contain additional information, in conjunction with the output commands %DISASSEMBLE, %DISPLAY, %HELP, %SDUMP and %TRACE.

– With *target-cmd* you specify the output command for which you want to define *medium-a-quantity.*

– With *medium-a-quantity* you specify which output media are to be used and whether or not additional information is to be output.

Command	Operand
%OUT	[target-cmd [medium-a-quantity][,...]]

In the %DISPLAY and %HELP commands, you may specify a separate *medium-a-quantity* operand which temporarily disables the declarations of the %OUT command for these commands. %DISASSEMBLE and %TRACE include no *medium-a-quantity* operand of their own; their output can only be controlled with the aid of the %OUT command.

Before selecting an output file with the %OUT command, the file must be assigned to a link name with the aid of the %OUTFILE command; otherwise, AID will use the default file names (see %OUTFILE).

The declarations made with the %OUT command are valid until overwritten by a new %OUT command, or until /LOGOFF.

An %OUT command without operands uses the default value T=MAX for all *target-cmds*.

%OUT may only be specified as an individual command, i.e. it may not be part of a command sequence or subcommand.

The %OUT command does not alter the program state.

target-cmd

Designates the command for which the declarations are to apply. Any of the following commands may be specified:

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

medium-a-quantity

Specifies the output medium or media for *target-cmd* and determines whether or not AID is to output additional information.

If the *medium-a-quantity* operand has been omitted, the default value T=MAX applies for *target-cmd*.

```
medium-a-quantity-OPERAND - - - - - - - - - - - - - - - - - - - - - - - - - - - -

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

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

The *medium-a-quantity* operand is described in more detail in the AID Core Manual, chapter 7.

T Terminal output via SYSOUT
H Hardcopy output (includes terminal output; cannot be specified in conjunction with *T*)
Fn Output to a file with link name F0...F7
P Output to SYSLST
MAX Output with additional information.
MIN Output without additional information.

Examples

1. `%OUT %DISPLAY T=MIN, F1=MAX`
   Data output of the %DISPLAY command is to be directed to the terminal in abbreviated form, and in parallel to the file with link name F1 along with additional information.

2. `%OUT %DISPLAY`
   For the %DISPLAY command, this specifies that previous declarations for output of data are canceled and the default value T=MAX applies.

%OUTFILE

With the %OUTFILE command you can assign output files to AID link names F0 through F7 or close output files. You can write output of the commands %DISASSEMBLE, %DISPLAY, %HELP and %TRACE to these files by specifying the corresponding %OUT command or *medium-a-quantity* operand. If a file does not yet exist, AID will make an entry for it in the catalog and then open it.

– With *link* you select the link name for the file to be cataloged and opened or closed.

– With *file* you assign a file name to the link name.

Command	Operand
%OUTFILE	[link [= file]]

If you do not specify the *file* operand, this causes AID to close the file designated via *link*. In this way the intermediate status of the file can be printed during debugging.

An %OUTFILE command without operands closes all open AID output files. If you have not explicitly closed an AID output file using the %OUTFILE command, the file will remain open until /LOGOFF.

Without %OUTFILE, you have two options of creating and assigning AID output files:

1. Enter a /SET-FILE-LINK command for a link name *Fn* which has not yet been reserved. Then AID opens this file when the first output command for this link name is issued.
2. Leave the creation, assignment and opening of files to AID. AID then uses default file names with the format AID.OUTFILE.Fn in accordance with link name *Fn*.

The %OUTFILE command does not alter the program state.

link

Designates one of the AID link names for output files and has the format Fn, where n is a number with a value $0 \leq n \leq 7$.

The REP records generated with the %MOVE command are always written to the output file with link name F6. This name should not be simultaneously used for other output purposes (see also the %AID and %MOVE commands).

file

Specifies the fully-qualified file name with which AID catalogs and opens the output file. An %OUTFILE command without the *file* operand closes the file assigned to link name *Fn*.

%QUALIFY

With the %QUALIFY command you define qualifications or an address, which can then be referenced in the address operand of another command by prefixing a period.
This abbreviated format is practical whenever you want to repeatedly access addresses which are not located in the current AID work area or if an address is to be used repeatedly as the start address for byte offset (•).

– The *prequalification* operand is used to specify qualifications or an address to be referenced in subsequent commands by prefixing a period.

Command	Operand
%QUALIFY	[prequalification]

A *prequalification* defined with the %QUALIFY command applies until it is superseded by a %QUALIFY command with a new *prequalification* or canceled by a %QUALIFY command without operands, or until /LOGOFF.

When entering a %QUALIFY command, only the syntax of the command is checked. Whether a dump file has been assigned to the specified link name and whether the specified area qualifications have been loaded and the associated object structure list exists is not checked until subsequent commands are executed and the specifications from the *prequalification* are actually used for addressing.

The declarations of the %QUALIFY command are only used by subsequently entered commands. A new %QUALIFY command has no effect on the subcommands in a %CONTROLn, %INSERT or %ON command previously entered, even if the subcommands are not executed until later on.

The current setting for handling uppercase/lowercase characters (%AID LOW={ON|OFF}) must be taken into account when entering the %QUALIFY command to ensure that the *prequalification* produces the correct extensions in the address operands of subsequent commands.

The %QUALIFY command may only be entered as an individual command, it may not be included in a command sequence or subcommand.

The %QUALIFY command does not alter the program state.

prequalification

Defines a single or multiple qualification or the start address for byte offset. By prefixing a period in the address operands of subsequent AID commands the *prequalification* defined in the %QUALIFY command can be referenced.

```
prequalification-OPERAND - - - - - - - - - - - - - - - - - - - - - - - - - - -

 ⎧    ⎧VM⎫                                                     ⎧C=csect   ⎫ ⎫
 ⎪[E=⎨  ⎬][[•]CTX=context][[•]L=load-unit][[•]O=object-module][[•]⎨          ⎬]⎪
 ⎪    ⎩Dn⎭                                                     ⎩COM=common⎭ ⎪
 ⎨                                                                          ⎬
 ⎪                   ⎧ALET={X'f...f'| %nAR| %nG}⎫      ⎧V'f...f'       ⎫    ⎪
 ⎪[E={VM|Dn}] [[•]⎨                          ⎬] [[•]⎨               ⎬]   ⎪
 ⎩                   ⎩SPID=X'f...f'             ⎭      ⎩compl-memref   ⎭    ⎭
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

- Consecutive qualifications must be separated by a period. In addition, there must be a period between the final *qualification* and the following operand part.

E={VM | Dn}
: Designates virtual memory (VM) or a dump file with the link name *Dn* (see %BASE).

{ALET={X'f...f' | %nAR | %nG} | SPID=X'f...f'}
: Designates a data area.

CTX=context
: Designates a context.

[L=load-unit•][O=object-module•]
: Designates the load unit and/or the object module.

{C=csect | COM=common}
: Designates a CSECT or a COMMON.

V'f...f' Designates a virtual address which can be predefined as the start address for byte offset.

compl-memref

Designates a calculated start address for a byte offset.
The following operations may occur in *compl-memref* (see section 7.2.4 in the AID Core Manual [1]):

- byte offset (•)
- indirect addressing (->)
- type modification (%A, %S, %SX)
- length modification (%L(...), %L=(expression), %Ln)
- address selection (%@(...))

Examples

1. ```
   %QUALIFY E=VM.L=MOD
   %DISPLAY .C=CS
   ```
   In the %DISPLAY command the string E=VM.L=MOD is inserted before the leading period, i.e. the CSECT with the name CS in load module MOD is output from the memory area of the loaded program. The qualification E=VM is required only if the current AID work area is in a dump file.

2. ```
   %QUALIFY V'100'
   %MOVE .0 INTO .#'A0'
   ```
   Without a preceding %QUALIFY command the %MOVE command would have the following appearance:
   ```
   %MOVE V'100'.0 INTO V'100'.#'A0'
   ```

3. ```
   %QUALIFY C=CSECT001.16
   %D .24
   ```
   Without a preceding %QUALIFY command the %DISPLAY command would be:
   ```
   %D C=CSECT001.16.24
   ```

4. ```
   %BASE E=VM
   %QUALIFY E=D1.%1G->
   %D .0 %FL4
   ```
   When the %DISPLAY command is executed the contents of AID register %1G are used as the address in the dump, the byte offset is added to this, and four bytes interpreted as an integer are output from the dump starting at this calculated address.

%REMOVE

With the %REMOVE command you revoke the test declarations for the %CONTROLn, %INSERT and %ON commands.

– With *target* you specify for which commands or command parts the declarations are to be deleted.

Command	Operand
%REM[OVE]	target

The %REMOVE command does not alter the program state.

target

Designates a command for which all the valid declarations are to be deleted, or a *test-point* to be deleted, or an *event* which is no longer to be monitored, or the subcommand to be deleted. If *target* is within a nested subcommand and therefore has not yet been entered, it cannot be deleted either.

```
target-OPERAND  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

{%C[ONTROL] | %C[ONTROL]n}
{%IN[SERT] | test-point  }
{%ON | %WRITE | event    }
{%•[subcmdname]          }

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

%C[ONTROL]
: The declarations for all %CONTROLn commands entered are deleted.

%C[ONTROL]n
: The %CONTROLn command with the specified number ($1 \leq n \leq 7$) is deleted.

%IN[SERT]
: All test points which have been entered are deleted.

test-point
: The specified *test-point* is deleted. *test-point* is specified as for the %INSERT command.
Within the current subcommand, *test-point* can also be deleted with the aid of %REMOVE %PC->, as the program counter (%PC) contains, at this point in time, the address of the test point.

%ON
: All events which have been entered are deleted.

%WRITE
: The *write-event* is deleted.

event
: The specified *event* is deleted. *event* is specified using a keyword, as with the %ON command. The *event* table with keywords and explanations of the individual events is listed under the %ON command.

  The following applies for the events %ERRFLG(zzz), %SVC(zzz) and %LPOV(zzz)
  %REMOVE *event(zzz)* deletes only the event with the specified number.
  %REMOVE *event* without specification of a number deletes all events of the corresponding group.

%•[subcmdname]
: Deletes the subcommand of a %CONTROLn or %INSERT command with *subcmdname*.

  %• is the abbreviated form for a subcommand name and can only be used within the subcommand. %REMOVE %. results in deletion of the subcommand which was most recently executed. This specification is only feasible as the last command in a subcommand, as subsequent commands under *subcmd* are not executed. As the %CONTROLn command cannot be chained, the associated %CONTROLn command is deleted too. Deletion of the subcommand thus corresponds to deletion of the %CONTROLn command with specification of the number.

  Multiple subcommands, however, may be chained at a *test-point* of the %INSERT command. Using %REMOVE %.[subcmdname], the user can delete an individual subcommand from a string, while further subcommands for the same *test-point* are retained (see AID Core Manual, chapter 5). If only the subcommand with *subcmdname* was entered for *test-point*, the *test-point* is deleted as well.

  %REMOVE %.[subcmdname] is not permitted in conjunction with %ON.

Examples

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

   Both %REMOVE commands have the same effect: %C1 is deleted.

2.
```
%IN V'100' <SUB1: %D %14, %15>
%IN V'100' <SUB2: %D %PC; %REM %.>
...
%REM V'100'
```

   When test point V’100’ is reached, the program counter is output, after which subcommand SUB2 is deleted. This subcommand is therefore executed once only. Then registers 14 and 15 are output and the program is continued. Whenever the program run then reaches test point V’100’, subcommand SUB1 is executed. The test point is deleted later using %REM V’100’. A %REM %.SUB1 command would have the same effect, since this subcommand is now the only one entered for test point V’100’.

%RESUME

With %RESUME you start the loaded program or continue it at the interrupt point. The program executes without tracing.

If the program has been interrupted during execution of a %TRACE command, the %TRACE command will be aborted. An interrupted %TRACE can only be continued by means of the %CONTINUE command.

Command	Operand
%R[ESUME]	

If a %RESUME command is contained within a command sequence or a subcommand, any commands which follow will not be executed.

If the %RESUME command is the only command in the subcommand, the execution counter is incremented and any active %TRACE deleted.

The %RESUME command alters the program state.

%SDUMP

In machine-oriented debugging, %SDUMP outputs the current call hierarchy.

– *%NEST* instructs AID to output the program names of the current call hierarchy.

– *medium-a-quantity* defines the output media to be used by AID and whether additional information is to be provided. This operand can be used to deactivate a declaration made with %OUT for the current %SDUMP.

Command	Operand	
%SD[UMP]	%NEST	[medium-a-quantity][,...]

In order to use this command, an ESD or ESV listing must be generated beforehand when compiling the program, and an object structure list or an external symbol dictionary should already have been created from it. Furthermore, the module AIDIT0 must have been loaded by the runtime system. This can be verified with %DISPLAY C=AIDIT0.

%NEST

The keyword %NEST initiates the output of the current call hierarchy. The output of AID differs considerably in this case from the corresponding output of %SDUMP %NEST when LSD information has been loaded. The source references, for example, are omitted.

medium-a-quantity

Defines the output medium or output media to be used and determines whether any supplementary information is to be shown in addition to the current call hierarchy.

If this operand is omitted, and no declaration with the %OUT command is in effect, AID will use the default value T=MAX.

```
medium-a-quantity-OPERAND - - - - - - - - - - - - - - - - - - - - - - - - - - - -

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

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

medium-a-quantity is described in detail in chapter 8 of the AID Core Manual [1].

T	Terminal output
H	Hardcopy output (includes terminal output; cannot be specified in conjunction with *T*)
Fn	File output
P	Output to SYSLST
MAX	Output with additional information.
MIN	Output with no additional information.

Example

The following sample listings show the call hierarchy of a C++ program. The first case reflects the output when the LSD information has been loaded. The library containing the LSD information is then disconnected for the current work area, and the call hierarchy is output again.

```
/STOPPED AT SRC_REF: 122, SOURCE: VPTR1.C , PROC: 7O::A_l::f(void)
/%sd /%nest
*** TID: 0028024F *** TSN: 0308 ************************************************
/SRC_REF: 122  SOURCE: VPTR1.C  PROC: 7O::A_l::f(void)  ************************
/SRC_REF: 127  SOURCE: VPTR1.C  PROC: vptr1_main(int)  *************************
/SRC_REF: 105  SOURCE: BCLB.C  PROC: main  *************************************
/ABSOLUT: V'17432'     SOURCE: ICS$MAI@  PROC: ICS$MAI@  ************************
```

```
/%symlib
/%sd /%nest
/ABSOLUT: V'534A'      SOURCE: VPTR1&@  PROC: f  ******************************
/ABSOLUT: V'58E0'      SOURCE: VPTR1&@  PROC: vptr1_main  *********************
/ABSOLUT: V'754'       SOURCE: BCLB&@  PROC: main  ****************************
/ABSOLUT: V'17432'     SOURCE: ICS$MAI@  PROC: ICS$MAI@  ************************
```

%SET

With the %SET command you transfer memory contents or AID literals to memory locations in the program which has been loaded. Before transfer, the storage types of *sender* and *receiver* are checked for compatibility. The contents of *sender* are matched to the storage type of *receiver*.

– With *sender* you designate the memory location to be transferred, a length, an address, a keyword, or an AID literal. *sender* may be either within the virtual memory of the loaded program or in a dump file.

– With *receiver* you designate the memory location to be overwritten. *receiver* may only be located within the virtual memory of the program which has been loaded.

Command	Operand
%S[ET]	sender INTO receiver

In contrast to the %MOVE command, the %SET command causes AID to check (prior to transfer) whether the *receiver* storage type is compatible with that of *sender* and whether the contents of *sender* match its storage type. In the event of incompatibility, AID rejects the transfer and outputs an error message.

If *sender* is longer than *receiver*, it is truncated at the left or right, depending on its storage type, and AID issues a warning. *sender* and *receiver* may overlap.

For numeric transfer, *sender* is converted to the storage type of *receiver* if required, and the contents of *sender* are transferred to *receiver* with the value retained. If the value of *sender* does not fully fit into *receiver*, AID issues a warning.

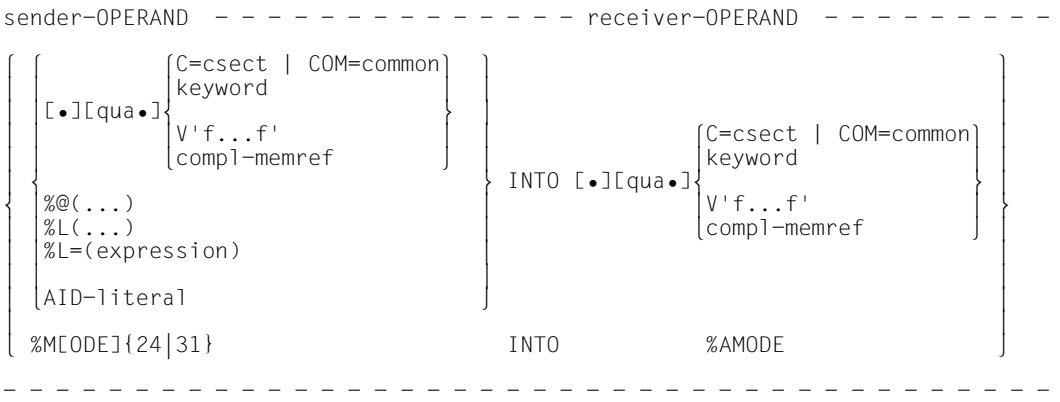
By entering the %AID CHECK=ALL command the user can activate an update dialog for checking purposes, which shows the old and new memory contents prior to execution of the transfer, providing the possibility of aborting the %SET command.

The %SET command does not alter the program state.

For *sender* and *receiver* the user may specify a virtual address, a complex memory reference or a keyword for general, floating-point and AID registers, the program counter or the execution counter. AID literals, selectors, and the keyword %MODEn can only be specified

as *sender*. Note that moving or overwriting instruction code may have undesirable side-effects if *sender* or *receiver* lies in a *control-area* or *trace-area* that was declared symbolically (see section 12.2 in the AID Core Manual [1]).

sender may be located either in virtual memory or in a dump file; *receiver*, on the other hand, may only be located in virtual memory.



- If the period is in a leading position it identifies a *prequalification*, which must have been defined by a preceding %QUALIFY command. A period must be inserted between consecutive qualifications. In addition, a period must be located between the last *qualification* and the subsequent operand part.

qua One or more qualifications may be specified here if a memory object to be addressed is not located in the current AID work area or is not unique in that area. It is sufficient to specify only the qualifications needed for a unique address.

{E={VM | Dn} for *sender* | E=VM for *receiver*}
A base qualification is needed only if the current base qualification is not to apply. *sender* can be either in virtual memory or in a dump file. *receiver*, on the other hand, can only be located in virtual memory.

{ALET={X'f...f' | %nAR | %nG} | SPID=X'f...f'}
Specified to reference an address in a data area, but only required when debugging ESA programs.

CTX=context
Specified only to enable unique addressing of a CSECT or a COMMON in the program system.

[L=load-unit•][O=object-module•]
Specified only if a CSECT or COMMON to be referenced cannot be uniquely addressed in the current context without these qualifications. You only need to specify the qualifications required for a unique address.

{C=csect | COM=common}
If addressing ends with a C/COM qualification, the entire selected program segment is transferred (*sender*), or overwritten from the beginning (*receiver*).

V'f...f' Specifies a virtual address.

compl-memref
The following operations may occur in *compl-memref* (see section 7.2.4 in the AID Core Manual [1] and chapter 4 in this manual for restrictions on debugging ESA programs):

- byte offset (•)
- indirect addressing (->)
- type modification (%T(dataname), %X, %C, %P, %D, %F, %A, %S, %SX)
- length modification (%L(...), %L=(expression), %Ln)
- address selection (%@(...))

An explicit type or length modification can be used to match the storage types of *sender* and *receiver*. Memory contents incompatible with a storage type are rejected by AID, however, even if a type modification is used.
For each operand in a complex memory reference, the associated memory area must not be exceeded as the result of a byte offset or length modification, otherwise AID will not execute the command and will output an error message.

If LSD records exist, the data names and statement names defined in the source program may also be used within *compl-memref*.

keyword

Designates a register, the program counter or an execution counter. Chapter 9 of the AID Core Manual [1] describes the implicit storage types of the keywords.

keyword	Meaning
%PC	Program counter
%n	General register, 0 ≤ n ≤ 15
%nD|E	Floating-point register, single/double precision, n = 0,2,4,6
%nQ	Floating-point register, quadruple precision, n = 0,4
%nG	AID general register, 0 ≤ n ≤ 15
%nDG	AID floating-point register, n = 0,2,4,6
%•[subcmdname]	Execution counter. The abbreviation %. designates the execution counter of the currently active subcommand.

In addition, the keywords for storage classes may be used (see section 10.5 in the AID Core Manual [1]).

%@(...)

With the address selector, you can use the start address of a CSECT or a memory area (%CLASS6, %CLASS6BELOW, %CLASS6ABOVE) or *compl-memref* as *sender*. The address selector supplies an address constant that can be used in a complex memory reference before a pointer operator (->). See also section 7.2.4.4 in the AID Core Manual [1]).

%L(...)

With the length selector, you can use the length of a CSECT or a memory area (%CLASS6, %CLASS6BELOW, %CLASS6ABOVE) as *sender*. The length selector supplies an integer value, which you can use for byte offset calculation or in a length function (see section 7.2.2.4 in the AID Core Manual [1].
Example: With %L(C=CS1) the length of CSECT CS1 is used as *sender*.

%L=(expression)

With the length function you can calculate the value of *sender*. *expression* consists of the contents of memory references, constants, integers and arithmetic operators. Only memory reference contents of type %F or %A with a length less than or equal to 4 are permitted. The length function supplies an integer, which can be used for byte offset calculation, in a further length function or for length modification (see AID Core Manual, chapter 6).
Example: With %L=(%1) the contents of register 1 are used as *sender*.

AID-literal

All AID literals listed in the AID Core Manual, chapter 8, may be specified (note the matching of AID literals to the respective *receivers* described there).

AID literal	Meaning
{C'x...x' | 'x...x'C | 'x...x'} X'f...f' | 'f...f'X} {B'b...b' | 'b...b'B} [{±}]n #'f...f' [{±}]n.m [{±}]mantissaE[{±}]exponent	Character literal Hexadecimal literal Binary literal Integer Hexadecimal number Decimal number Floating-point number

%M[ODE]{24|31}

Defines the addressing mode.

%AMODE

Is the system information field in which the addressing mode is entered.

Transfer of keyword %M{24|31} to the system information field %AMODE changes the addressing mode of the test object. If only the address interpretation for indirect addressing (->) is to be modified, the corresponding %AINT command should be entered.

If the program counter is to be modified when testing a 31-bit-address program while 24-bit mode is switched on, care must be taken to ensure that the most significant byte has the value X’00‘.

SET table

The following table shows how AID effects transfer, how the various storage types are converted, and which transfers are rejected.

sender	*receiver*		
	%F %P %A %n %nG %PC %.[name] %D %nE/D/Q %nGD	%C	%X
%F %P %A {±}n #'f...f' %n %nG %PC %.[name] %D %nE/D/Q %nGD	num	–	bin
{±}n.m {±}mantE{±}exp	num	–	–
%C C'x...x'	num(1)	char	bin
%X X'f...f' B'b...b'	bin	bin	bin

bin — Binary transfer, left-justified
sender < *receiver*: padded with binary zeros on the right
sender > *receiver*: truncated on the right
For transfer to storage type %X a numeric literal corresponds to a signed integer with a length of four bytes (%FL4), which are transferred in binary form.

char — Character transfer, left-justified
sender < *receiver*: padded with blanks (X'40') on the right
sender > *receiver*: truncated on the right

num(1) — If a character *sender* consists of digits only and is no more than 18 characters in length, it is transferred in numeric form if *receiver* is of the numeric type; otherwise AID rejects transfer.

num — Numeric transfer with value retained
sender is matched to the storage type of *receiver* if required.

– — No transfer
AID issues a message as to incompatibility of the storage types.

Examples

1.

```
/%D V'798'
V'00000798' = M1BS     + #‘00000798'
00000798 (00000798) 012CF000                                  ..0.
/%SET V'798'%PL2 INTO %1G
/%D %1G, %1G%F
%1G             = 0000000C
%1G             =          12
```

The contents of the first two bytes at address V’798’ are interpreted as a packed number and transferred to AID register %1G. The AID register is of type %F (binary with sign), the value 12 is correspondingly converted.

2.

```
/%SET  V'300'%PL1 INTO %1
/%SET  V'400'%PL1 INTO %2
/%DISPLAY V'300', V'400'
V'00000300' = M1BS     + #‘00000300'
00000300 (00000300) 1C010080                                  ....
V'00000400' = M1BS     + #‘00000400'
00000400 (00000400) 2D000000                                  ....
/%DISPLAY %1, %2
%1              = 00000001
%2              = FFFFFFFE
```

The two packed contents at addresses V’300’ and V’400’ correspond to the values +1 and -2. The values are converted accordingly during transfer to a register.

3.

```
/%DISPLAY V'100', V'200'
V'00000100' = M1BS     + #‘00000100'
00000100 (00000100) F0000000                                  0...
V'00000200' = M1BS     + #‘00000200'
00000200 (00000200) 01020000                                  ....
/%SET  V'100'%CL1 INTO V'101'%CL2
/%SET  V'200'%XL2 INTO V'202'%XL4
/%DISPLAY V'100', V'101'%CL2, V'200', V'202'%XL4
V'00000100' = M1BS     + #‘00000100'
00000100 (00000100) F0F04000                                  00 .
V'00000101' = M1BS     + #‘00000101'
00000101 (00000101) 0
V'00000200' = M1BS     + #‘00000200'
00000200 (00000200) 01020102                                  ....
V'00000202' = M1BS     + #‘00000202'
00000202 (00000200) 01020000                                  ....
```

For transfer in type %C (character), the *receiver* is padded with blanks (X’40’) on the right. For transfer in type %X (hexadecimal), the *receiver* is padded with binary zeros (X’00’) on the right.

4.

```
/%SET  %1 INTO V'100'%D
/%DISPLAY V'100', V'100'%D
V'00000100' = M1BS     + #'00000100'
00000100 (00000100) 41100000                                  ....
V'00000100' = M1BS     + #'00000100'
00000100 (00000100)+.10000000000000000000000 E+001
```

Register %1 contains the value 1 (X'00000001'), which is converted to storage type %D (floating point) and transferred to address V'100'.

5.

```
/%SET X'00' INTO V'100'%L30
```

In contrast to %MOVE, *sender* is extended in %SET to the length of *receiver* in accordance with its data type, so 30 bytes as of the address V '100' are overwritten with the value 'hexadecimal zero'.

%SHOW

The %SHOW command allows the user to obtain information about the current definitions relating to individual AID commands, to find out what the last entry of a command looked like, and which command was entered last. It is also possible to use the subcommand name to request the command in which it was defined or to output a list of all entered subcommand names with the associated command type. Depending on how uppercase and lowercase notation was defined in the %AID command, the original entry of the command is either reproduced or the input string is converted to uppercase letters.

- *show-target* can be used to specify a command, a subcommand name or an AID keyword for all current subcommands.

Command	Operand
%SH[OW]	[show-target]

The effect of %SHOW without an operand is to output the AID command entered directly beforehand. If no AID command has been entered for the task, an error message is issued. A %SHOW for one of the commands for which it is not intended results in a syntax error. The command may be used in command and subcommand strings.

%SHOW does not alter the program state.

show-target

designates an AID command, a specific subcommand or all entered subcommands. The commands permitted for this command can also be specified in the abbreviated form in *show-target*.

Command or subcommand	Information
%AID	The current valid settings for the %AID, %AINT and %BASE commands and the version of AID loaded.
%BASE	The current settings for %BASE, %AINT and %SYMLIB, the TSN, TID and the version of the operating system and type of computer are output.
%C[ONTROL]	The input string is output for each registered %CONTROL*n*.
%D[IS]A[SSEMBLE]	The current *number* and ***start*** address (V'...') is output.
%F[IND]	The entered command and, if appropriate, the virtual address of the last hit are output.
%IN[SERT] [test-point]	Without the ***test-point*** entry, all active test points are output. Otherwise, AID shows the entered command in which *test-point* was declared.
%ON	The input string is output for each active %ON command.
%OUT	The valid ***medium-a-quantity*** values for the commands that can be controlled via %OUT are output.
%OUTFILE	All implicitly or explicitly entered output files are listed, with their link names.
%QUALIFY	The last %QUALIFY command is output.
%SYMLIB	The registered libraries are output with the associated base qualifications and the TSN.
%TRACE	The default values of the %TRACE operands are output, taking into account whether the last %TRACE was symbolic or on machine code level. In trailing lines AID shows how many instructions or statements have already been processed with the current %TRACE and what the input string of the last %TRACE command looked like.
%.*	The names of all active subcommands are output with the type of the AID command in which they were defined.
%.subcmdname	The command in which ***subcmdname*** was defined is output.

%STOP

With the %STOP command you direct AID to interrupt the program, to switch to command mode and to issue a STOP message. This message indicates the address and the CSECT where the program was interrupted.

If the command is entered at the terminal or from a procedure file, the program state is not altered, since the program is already in the STOP state. In this case you may employ the command to obtain localization information on the program interrupt point by referring to the STOP message.

Command	Operand
%STOP	

If the %STOP command is contained in a command sequence or subcommand, any commands following it will not be executed.

If the program has been interrupted by pressing the K2 key, the program interrupt point is not necessarily within the user program but may also be located in the runtime system routines.

The %STOP command alters the program state.

%TITLE

With the %TITLE command you define the text of your own page header. AID uses this text when the %DISASSEMBLE, %DISPLAY, %HELP, %SDUMP and %TRACE commands write to the system file SYSLST.

– By means of the *page-header* operand you specify the text of the header, directing AID to set the page counter to 1 and position SYSLST to the top of the page before the next line to be printed.

Command	Operand
%TITLE	[page-header]

With a %TITLE command without a *page-header* operand you switch back to the AID standard header. AID again sets the page counter to 1 and positions SYSLST to the top of the page before the next line to be printed.

The %TITLE command does not alter the program state.

page-header

Specifies the variable part of the page title. AID completes this specification by adding the time, date and page count.

page-header
: is a character literal in the format {C'x...x' | 'x...x'C | 'x...x'} and may have a maximum length of 80 characters. A longer literal is rejected with an error message outputting only the first 52 positions of the literal.

  Up to 58 lines are printed on one page, not counting the title of the page.

%TRACE

With the %TRACE command you switch on the AID tracing function and start the program or continue it at the interrupt point.

– By means of the *number* operand you define the maximum number of instructions to be traced, i.e. executed and logged.

– By means of the *criterion* operand you select different types of machine instructions. AID outputs a logging line after an instruction of the selected type has been executed.

– By means of the *trace-area* operand you define the program area in which the *criterion* is to be taken into consideration.

Command	Operand		
%T[RACE]	[number]	criterion	[IN trace-area]

A %TRACE cannot be active in conjunction with a *write-event* of %ON.

On machine code level the %TRACE command has a **different effect** than on the symbolic level.
In debugging on machine code level, an instruction selected with *criterion* is logged following its execution and **monitored also outside** of the *trace-area*. Thus *trace-area* only has an effect on logging, not on monitoring. The additional monitoring effort caused by this slows down the program run; therefore insertion of the %TRACE command as a subcommand of the %INSERT command directly at test points and its deletion immediately after execution are recommended for large programs with long instruction tracts (see chapter 6 in the AID Core Manual [1] and the %INSERT command).

The %TRACE command is **interrupted** by the following events during the test run:

– A subcommand containing a %STOP is executed.
– An %INSERT command terminates with a program interrupt, as the *control* operand is K or S.
– The K2 key is pressed. For more information, see the section 3.2, “Commands at the beginning of a debugging session” on page 8.

The %TRACE command is **terminated** by the following events:

– The maximum number of instructions to be traced is reached (which is why a %TRACE can be deleted by entering %T 1 %INSTR).
– You enter a %RESUME after one of the program interrupts described above.
– A subcommand containing a %RESUME or %TRACE command is executed.

If a %TRACE command has been terminated as described above, its operand values still remain valid. They continue to apply until they are superseded by specifications in a new %TRACE command or until end of program. AID therefore inserts the value from the previous %TRACE command in a new %TRACE command in which an operand has not been specified. For the *trace-area* operand, this is done only if the current interrupt point is situated in the *trace-area* to be taken over. If there are no values to be taken over, AID assumes 10 (for *number*) and the entire user program (for *trace-area*) by default.

With the aid of the %OUT command, you can control the information to be contained in a line of the log and the output medium to which the log is to be directed.

If the %TRACE command is contained in a command sequence or subcommand, any commands which follow will not be executed.

The %TRACE command alters the program state.

number

Specifies the maximum number of instructions of type *criterion* which are to be executed and logged.

number
is an integer where $1 \leq number \leq 2^{31}-1$.
The default value is 10. If there is no value from a previous %TRACE command, AID inserts the default value in a %TRACE command with no *number* operand.

After the specified *number* of instructions have been traced, AID outputs a message via SYSOUT, the program is interrupted and the user can enter AID or BS2000 commands. The message tells you at which address and in which CSECT or COMMON the program was interrupted.

criterion

This is a keyword which defines the type of machine instructions to be traced after execution. The default value is the **symbolic** *criterion* %STMT, so a *criterion* must always be specified for machine-oriented debugging unless a suitable *criterion* declaration from a previous %TRACE command is still valid.

criterion	Logging takes place after execution of:
%INSTR	each machine instruction executed
%B	each branch instruction executed (these are the machine instructions BAL, BALR, BAS, BASSM, BASR, BC, BCR, BCT, BCTR, BSM, BXH and BXLE)
%BAL	each invocation of a subprogram (via machine instructions BAL, BALR, BAS, BASSM and BASR).

trace-area

Defines the program area in which tracing is to take place.
The base qualification E=VM must be set (see %BASE) or specified explicitly for %TRACE.

A *trace-area* definition remains effective until a new %TRACE command with its own *trace-area* operand is entered,
until a %TRACE command is issued outside of this area
or until the end of the program.
If the *trace-area* operand has been omitted, the area definition from an earlier %TRACE command is assumed if the current interrupt point is located in this area. Otherwise AID assumes the entire user program by default.

criterion continues to be monitored outside of the *trace-area*, but in this case the commands which have been executed are not logged. Due to the resulting additional monitoring expenditure, use of the %TRACE command directly at test points and subsequent deletion are recommended for large programs (see %INSERT).

```
trace-area-OPERAND  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

                    ⎧ C=csect | COM=common   ⎫
IN [•][qua•] [•]    ⎨ ( V'f...f': V'f...f' ) ⎬
                    ⎩ keyword                ⎭

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

- If the period is in the leading position it denotes a *prequalification*, which must have been defined with a preceding %QUALIFY command. Consecutive qualifications must be separated by a period. In addition, there must be a period between the last *qualification* and the following operand part.

qua One or more qualifications may be specified here if a memory object to be addressed is not located in the current AID work area or is not unique in that area. It is sufficient to specify only the qualifications needed for a unique address.

E=VM
Since *trace-area* can only be located in virtual memory of the loaded program, the base qualification *E=VM* is only required if *E=Dn* was declared.

CTX=context
Specified only to enable unique addressing of a CSECT or a COMMON in the program system.

[L=load-unit•][O=object-module•]
Specified only if a CSECT or COMMON to be referenced cannot be uniquely addressed in the current context without these qualifications. You only need to specify the qualifications required for a unique address.

{C=csect | COM=common}
The *trace-area* includes the entire CSECT or COMMON that was specified.

(V'f...f' : V'f...f')
The *trace-area* is defined by specifying a virtual start address and a virtual end address. The addresses must be located in the executable part of a loaded program. *f...f* is a valid hexadecimal address with up to eight digits. The start address must be less than or equal to the end address.

keyword
The *trace-area* covers the memory area addressed by one of the following keywords (see chapter 9 in the AID Core Manual [1]). In the case of XS and ESA systems, %CLASS6 and %CLASS6BELOW designate the same address area.

keyword	Meaning
%CLASS6	Class 6 memory
%CLASS6BELOW	Class 6 memory below the 16-Mb limit
%CLASS6ABOVE	Class 6 memory above the 16-Mb limit

Output of the %TRACE listing

By default the %TRACE listing is output in extended form via SYSOUT (T=MAX). The %OUT command can be used to define the output media and determine how AID is to format the output lines (see chapter 7 in the AID Core Manual [1]).

When debugging ESA programs, addresses located in a data area are marked in the %TRACE listing by an asterisk (*). In addition, the access registers (ARn=...) are shown.

An extensive %TRACE listing (%OUT parameter T=MAX) includes the following information (point 1 and point 4 only with output to SYSLST):

1. CSECT-relative address of the machine instruction
2. Instruction in Assembler notation
3. Condition code
4. Various information on the operands of an instruction, depending on the instruction type:
   - current operand values
   - for an SVC, the associated macro name and the parameter list (up to 12 bytes)
   - a conditional branch instruction is flagged by an asterisk (*) in front of the mask field operand if the condition is satisfied, i.e. if the branch has been executed (if the mask field contains binary zeros, the string "NOP" is output instead of the mask image)
   - the EX instruction and the instruction executed by it are logged in the same format
   - address operands are output with the calculated end address, which is additionally shown as a CSECT-relative address.

The following additional information is included in the output to:
SYSLST (%OUT %T P=MAX):

5. Virtual address of the machine instruction
6. Instruction in hexadecimal format

Example of line format (T=MAX)

```
M1BS+C          BALR  R15,R0                3 R15=7F00000E  R0=00000000
M1BS+E          LM    R2,R13,56(R15)         3  A2=00000064=M1BS+64
                                                R2=000008A0  R3=00000798
                                                R4=00000000  R5=00000000
                                                R6=00000000  R7=00000000
                                                R8=00000000  R9=00000000
                                               R10=00000000 R11=000000A8
                                               R12=000069D0 R13=00000A00
M1BS+12         LA    R0,5(R0,R0)            3  R0=00000005
                                                A2=00000005=M1BS+5
M1BS+16         LA    R1,3E(R0,R15)          3  R1=0000004C
                                                A2=0000004C=M1BS+4C
M1BS+1A         L     R15,94(R0,R11)         3 R15=00003868
                                                A2=0000013C=M1BS+13C
                                                O2=00003868
```

%OUT %T T=MIN produces a different output format. Items 1 and 6 are not included, but items 2, 3, 4 and 5 are always output, regardless of the type of output medium.

Example of line format (T=MIN)

```
0000000C  BALR  R15,R0             3 R15=7F00000E  R0=00000000
0000000E  LM    R2,R13,56(R15)     3  A2=00000064  R2=000008A0  R3=00000798
                                      R4=00000000  R5=00000000  R6=00000000
                                      R7=00000000  R8=00000000  R9=00000000
                                     R10=00000000 R11=000000A8 R12=000069D0
                                     R13=00000A00
00000012  LA    R0,5(R0,R0)        3  R0=00000005  A2=00000005
00000016  LA    R1,3E(R0,R15)      3  R1=0000004C  A2=0000004C
0000001A  L     R15,94(R0,R11)     3 R15=00003868  A2=0000013C  O2=00003868
```

Examples

1. `%T 3 %INSTR`
   The next three instructions in the current CSECT are traced. The program is then interrupted, and a STOP message and END-OF-TRACE message are output.

2. `%T`
   The operand values of the preceding %TRACE are assumed for this %TRACE. The program is continued, and three further instructions are traced.

3. `%T 10 %B IN E=VM.(V'83E4':V'8488')`
   If a base qualification applies to a dump file, the base qualification E=VM must be specified for the *trace-area*. A maximum of 10 branch instructions are traced in the program range V'83E4' to V'8488'. If 10 instructions have not been logged in the first pass of *trace-area*, the %TRACE remains active, and the branch instructions continue to be monitored, even though they are not logged. This slows down the program. The next example provides a solution to the problem.

4. `%INSERT V‘A38‘ <%T 20 %B IN (V‘A38‘:V‘B40‘)>`
   `%INSERT V‘B40‘ <%R>`
   From address V’A38’ to V’B40’, a trace line is to be output after every branch instruction. The %TRACE command does not become active, however, until the program run reaches address V’A38’ (%INSERT). When the *trace-area* is exited via address V‘B40‘, the subcommand of the second %INSERT deletes the %TRACE. This example is based on the assumption that the trace area has only one entry and exit.

6 Sample application

This chapter illustrates an AID debugging session for a small Assembler program. With this debugging session as an example you can simulate the use and effect of a good number of AID commands; the method of proceeding we selected is simple, for the sake of clarity. The Assembler program is given in section 6.1, the debugging sequence in section 6.2. Entries to be made are printed in bold for better legibility.

6.1 Assembler program

Objective

The program named TOTAL reads in 10 two-digit numbers and outputs the total. Input of the number 00 serves as the end criterion. If more than 10 numbers are entered, a message is issued and the calculated total is output.

Excerpt from the assembly listing

```
CALCULATION OF THE TOTAL OF N NUMBERS (N <= 10)                                                    14:09:30
1995-02-21
          SYMBOL   TYPE ID  ADDR     LENGTH  A/R-MODE                EXTERNAL SYMBOL DICTIONARY
          TOTAL     SD 0001 00000000 000184  ANY ANY
CALCULATION OF THE TOTAL OF N NUMBERS (N <= 10)                                                    14:09:30
1995-02-21
 LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
 000000                                       1     TOTAL    START
                                              2              TITLE 'CALCULATION OF THE TOTAL OF N NUMBERS (N <= 10)'
                                              3              PRINT NOGEN
                         00000000             4     R0       EQU   0
                         00000001             5     R1       EQU   1
                         00000002             6     R2       EQU   2
                         00000003             7     R3       EQU   3
                         00000004             8     R4       EQU   4
                         00000005             9     R5       EQU   5
                                             10     TOTAL    AMODE ANY
                                             11     TOTAL    RMODE ANY
                                             12              GPARMOD 31
                                             14   2                *,VERSION 010
 000000 0D 20                                15              BASR  R2,R0
 000002                   00000002            16              USING *,R2
 000002                                       17     BEGIN    WROUT MESG1,END
                                              46   2                *,@DCEO      999    921011   53531004
 000026 58 50 2176        00000178            49              L     R5,=F'1'
 00002A 5A 50 2176        00000178            50     LOOP     A     R5,=F'1'
 00002E 49 50 2138        0000013A            51              CH    R5,TEN
 000032 47 20 20BE        000000C0            52              BH    ERROR
 000036                                       53     READ     RDATA INPUT,END
                                              88   2                *,@DCEI      999    921011   53531002
 000062 D5 05 2121213A 00000123 0000013C      91     COMP     CLC   INPUT+4,NULL
 000068 47 80 207A        0000007C            92              BE    OUT
 00006C F2 11 21232121 00000125 00000123      93     ADD      PACK  PACK,INPUT+4(2)
 000072 FA 31 213C2123 0000013E 00000125      94              AP    SUM,PACK
 000078 47 F0 2028        0000002A            95              B     LOOP
 00007C F3 63 2131213C 00000133 0000013E      96     AUS      UNPK  RESLT,SUM
 000082 D3 00 21372140 00000139 00000142      97              MVZ   RESLT+6(1),ZONE
 000088                                       98              WROUT MESG2,END
                                             126   2                *,@DCEO      999    921011   53531004
 0000AA                                      129     END      TERM  DUMP=Y
                                             132   2                *,VERSION 100
 0000BE                                      144     ERROR    WROUT MESG3,END
                                             173   2                *,@DCEO      999    921011   53531004
 0000E2 47 F0 207A        0000007C           176              B     OUT
                                             177     *
                                             178     *
                                             179     * DEFINITIONS
                                             180     *
 0000E6 0039                                 181     MESG1    DC    Y(L'M1+5)
 0000E8 404001                               182              DC    X'404001'
 0000EB C2C9E3E3C540C2C9                     183     M1       DC    C'PLEASE ENTER UP TO 10 TWO-DIGIT NUMBERS! END:
00'
 00011F 000000000000                        184     INPUT    DC    XL6'00'
 000125 000C                                185     PACK     DC    PL2'0'
                                             186     *
 000128 0012                                 187     MESG2    DC    Y(L'M2+L'RESLT+5)
 00012A 404001                               188              DC    X'404001'
 00012D E2E4D4D4C57A                         189     M2       DC    C'TOTAL:'
 000133 40404040404040                       190     RESLT    DC    CL7' '
                                             191     *
 00013A 000A                                 192     ZEHN     DC    H'10'
 00013C F0F0                                 193     NULL     DC    C'00'
 00013E 0000000C                             194     GESAMT   DC    PL4'0'
 000142 F0                                   195     ZONE     DC    X'F0'
                                             196     *
 000144 0034                                 197     MESG3    DC    Y(L'M3+5)
 000146 404001                               198              DC    X'404001'
 000149 C5E240D2D6C5D5D5                     199     M3       DC    C'NO MORE THAN 10 NUMBERS CAN BE PROCESSED'
 000000                                      200              END   TOTAL
 000178 00000001                             201                    =F'1'
 00017C 95022114065611 83                    202                    =X'95022114065611 83' CONSISTENCY CONSTANT FOR AID
FLAGS IN 00000 STATEMENTS, 000 PRIVILEGED FLAGS, 000 MNOTES
HIGHEST ERROR-WEIGHT : NO ERRORS
THIS PROGRAM WAS ASSEMBLED BY ASSEMBHC    V 1.2A00     ON 1995-02-21 AT 14:09:30
CALCULATION OF THE TOTAL OF N NUMBERS (N <= 10)
```

6.2 Test run

Step 1

The Assembler source program TOTAL in file SOURCE.TEST is assembled using ASSEMBH. The source program is assembled without error.

```
/DEL-SYS-FILE OMF
/START-PROG ASSEMBH

%  BLS0500 PROGRAM 'ASSEMBH', VERSION '1.2A00' OF '1994-11-11' LOADED.
%  BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG. 1990. ALL RIGHTS
 RESERVED
%  ASS6010 V 1.2A00 OF BS2000 ASSEMBH  READY

//COMPILE SOURCE=SOURCE.TEST,MODULE-LIBRARY=LMSLIB

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

//END

%  ASS6012 END OF ASSEMBH
```

Step 2

The TOTAL program is to be executed.

```
/START-PROG *MOD(LIB=LMSLIB,ELEM=TOTAL)
%  BLS0517 MODULE 'TOTAL' LOADED

PLEASE ENTER UP TO 10 TWO-DIGIT NUMBERS! END:
*05
*16
*48
*00
*0
*00
*EN
*
/%STOP
STOPPED AT V'60' = TOTAL + #'60'
```

The program always branches back to input, which means that the end-of-input indicator ‘00’ is not recognized. The program is therefore interrupted by pressing the K2 key, and the %STOP command is used to output the address of the current interrupt point. This is located in the RDATA handling. It is followed by the check for end-of-input with the CLC instruction at address V’62’.

Step 3

```
/LOAD-PROG *MOD(LIB=LMSLIB,ELEM=TOTAL)

%  BLS0517 MODULE 'TOTAL' LOADED

/%SET #'62' INTO %1G
/%INSERT %1G-> <%D V'11F'%L6;%STOP>
/%R
```

The program is loaded again and a test point is set for the CLC instruction. The address at which the CLC instruction is located is stored in AID register %1G, since this address is often used. Each time the program run reaches this address, the contents of the field INPUT (address V'11F') are to be output. Subsequent to output, the program is to switch to the STOP state so that new commands may be entered.

The %RESUME command is used to start the loaded program.

Step 4

```
PLEASE ENTER UP TO 10 TWO-DIGIT NUMBERS! END:
*05

*** TID: 0001020D *** TSN: 2069 ***********************************************
CURRENT PC: 00000062    CSECT: TOTAL     ************************************
V'0000011F' = TOTAL    + #'0000011F'
0000011F (0000011F) 00060000 F0F5                          .... 05
STOPPED AT V'62' = TOTAL     + #'62'

/%R
*00
V'0000011F' = TOTAL    + #'0000011F'
0000011F (0000011F) 00060000 F0F0                          .... 00
STOPPED AT V'62' = TOTAL     + #'62'
```

Following the input of a normal number, the end-of-input indicator “00” is entered in response to the second prompt to enter a number.

Step 5

```
/%T 3 %INSTR
TOTAL+62          CLC   121(6,R2),13A(R2)      2  A1=00000123=TOTAL+123
                                                   A2=0000013C=TOTAL+13C
                                                   O1=F0F0055D 0000
                                                   O2=F0F00000 041D
TOTAL+68          BC    B'1000',7A(R0,R2)      2   M=0
TOTAL+6C          PACK  123(2,R2),121(2,R2)    2  A1=00000125=TOTAL+125
                                                   A2=00000123=TOTAL+123
                                                   O1=000F O2=F0F0
STOPPED AT V'72' = TOTAL + #'72' , END OF TRACE

/%D V'64'%S->%L6
V'00000123' = TOTAL    + #'00000123'
00000123 (00000123) F0F0005F 0000                          00.)..

/%D V'66'%S->%L6
V'0000013C' = TOTAL    + #'0000013C'
0000013C (0000013C) F0F00000 005C                          00...*
```

The check for end-of-input is now followed using %TRACE. When 3 Assembler instructions have been traced, the %TRACE terminates with a STOP message. The end-of-input indicator '00' was not recognized, and the program continued in the branch to "add the input value". The two following %DISPLAY commands output the memory contents compared in the CLC. The addresses are calculated directly from parts of the instruction by means of the type modification %S with the following pointer operator. The CLC instruction uses the implicit length 6 of the first address operand (INPUT), but the matching field NULL was only defined for 2 digits and the input also comprises only 2 digits, so no match can be recognized by the CLC instruction.

The correct Assembler command should read:

```
COMP       CLC        INPUT+4(2),NULL
```

Step 6

This error can be temporarily eliminated by using the %MOVE command. The program is loaded again, and the update dialog is activated with %AID.

```
/LOAD-PROG *mod(lib=bib,elem=TOTAL)
%  BLS0517 MODULE 'TOTAL' LOADED

/%AID CHECK=ALL
/%MOVE X'01' INTO %1G->.1

OLD CONTENT:
05
NEW CONTENT:
01
%  AID0274 Change desired? Reply (Y=Yes; N=No)?
/Y
/%R
```

The %MOVE command changes the length specification of the CLC instruction from ’05’ to ’01’. This is verified in an update dialog, and the program is then started with %RESUME.

```
*05
*16
*48
*12
*10
*15
*17
*19
*29
NO MORE THAN 10 NUMBERS CAN BE PROCESSED
TOTAL:0000171
```

Another program error has been found, as the user entered only nine numbers and not ten.

Step 7

```
/LOAD-PROG *MOD(LIB=BIB,ELEM=TOTAL)
%  BLS0517 MODULE 'TOTAL' LOADED

/%AID CHECK
/%M X'01' INTO %1G->.1
/%T 4 %INSTR IN (V'26':V'36')
PLEASE ENTER UP TO 10 TWO-DIGIT NUMBERS! END:
TOTAL+26          L     R5,176(R0,R2)          0  R5=00000001
                                                  A2=00000178=TOTAL+178
                                                  O2=00000001
TOTAL+2A          A     R5,176(R0,R2)          2  R5=00000002
                                                  A2=00000178=TOTAL+178
                                                  O2=00000001
TOTAL+2E          CH    R5,138(R0,R2)          1  R5=00000002
                                                  A2=0000013A=TOTAL+13A
                                                  O2=000A
TOTAL+32          BC    B'0010',BE(R0,R2)      1   M=2
                                                  A1=000000C0=TOTAL+C0
STOPPED AT V'36' = TOTAL + #'36' , END OF TRACE
```

%AID is used to deactivate the update dialog.
%MOVE is used to repeat the tentative correction.
%TRACE monitors and traces the program segment V'26' to V'36'. The %TRACE listing shows that register 5 was loaded with a value of 1 at the start and thus contained the value ‘2’ even before the first number was read. The correct instruction with address V’26’ should read:

```
L       R5,=F'0'
```

Step 8

This error can be temporarily removed by means of the %SET command. The program is then reloaded.

The correction of the CLC instruction with %MOVE is repeated, but this time with the correct address.

```
/LOAD-PROG *MOD(LIB=BIB,ELEM=TOTAL)
%  BLS0517 MODULE 'TOTAL' LOADED

/%M X'01' INTO V '63'
/%INSERT V‘2A‘ <%SET 0 INTO %5; %REM %INSERT>
/%R
```

%INSERT sets a test point after the incorrect load instruction with address V‘26‘. A %SET in the subcommand of %INSERT sets register 5 to ‘0’. The %INSERT is deleted with %REMOVE, and the program is then continued. An alternative solution would be to alter the load instruction directly (as shown earlier for the length in the CLC) so that register 5 is correctly loaded from the start. Since the INPUT field (address V’11F’) is loaded with X’00’, you could then specify:

```
%MOVE X‘211F‘ INTO V‘28‘
```

```
PLEASE ENTER UP TO 10 TWO-DIGIT NUMBERS! END:
*05
*16
*48
*12
*10
*15
*17
*19
*29
*11
NO MORE THAN 10 NUMBERS CAN PROCESSED
TOTAL:0000182
```

Once this correction has been made the program executes without errors, and permanent error recovery can be effected in the source program.

Glossary

Access register

%nAR, $0 \leq n \leq 15$. Access registers are available on ESA systems in parallel to multipurpose registers. They are required for addressing data spaces. An ALET (assess list entry token) referring to a data space via the access list is entered into them. If AR mode is activated, the access registers are evaluated at the same time as the addresses are converted.

addressing mode

The addressing mode determines how addresses are to be converted for the execution of machine instructions. By default, AID assumes the addressing mode of the object being debugged. This applies to the address length (24 or 31 bits) for programs running on XS computers (%AMODE) and also to the addressing of data areas on ESA systems (%ASC).

System information on the address length can be referenced with the keyword %AMODE. This setting can be checked with %DISPLAY and modified with `%MOVE %MODE{24|31} INTO %AMODE`.

The keyword %ASC (access space control mode) references the system information for the AR mode (access register mode). It returns information on whether access registers for addressing data areas are included in the address conversion. This setting can also be checked with %DISPLAY.

address operand

This is an operand used to address a memory location or a memory area. Virtual addresses, data names, statement names, source references, keywords, complex memory references, a C qualification (debugging on machine code level) or a PROG qualification (symbolic debugging) may be specified. The memory location/area is situated either in the loaded program or in a memory dump in a dump file. If a name has been assigned more than once in a user program and thus no unique address reference is possible, area qualifications or an *identifier* (COBOL) can be used to assign the name unambiguously to the desired address.

AID default address interpretation

Indirect addresses, i.e. addresses preceding a pointer operator, are interpreted according to the currently valid addressing mode of the test object by default. %AINT can be used to deviate from the default address interpretation and to define whether AID is to use 24-bit or 31-bit addresses in indirect addressing.

AID input files

AID input files are files which AID requires to execute AID functions, as distinguished from input files which the program requires. AID processes disk files only. AID input files include:

1. Dump files containing memory dumps (%DUMPFILE)
2. PLAM libraries containing object modules (OMs) or link and load modules (LLMs). If the library has been assigned with the %SYMLIB command, LSD records can be dynamically loaded by AID.

AID literal

AID provides the user with both alphanumeric and numeric literals (see chapter 9 in the AID Core Manual [1]):

```
C'x...x' | 'x...x'C | 'x...x'}          Character literal
{X'f...f' | 'f...f'X}                  Hexadecimal literal
{B'b...b' | 'b...b'B}                  Binary literal
[{±]n                                  Integer
#'f...f'                               Hexadecimal number
[{±}]n.m                               Decimal number
[{±}]mantissaE[{±}]exponent                Floating-point number
```

AID output files

These are files to which the output of the %DISASSEMBLE, %DISPLAY, %HELP, %MOVE, %SDUMP and %TRACE commands may be written. The files are referenced via their link names F0 through F7 (see %OUT and %OUTFILE).
The REP records are written to the file assigned to link name F6 (see %AID REP=YES and %MOVE).
There are three ways of creating an output file:

1. /%OUTFILE command with link name and file name
2. /FILE command with link name and file name
3. AID issues a FILE macro with the file name AID.OUTFILE.Fn for a link name to which no file name has been assigned.

An AID output file always has the format FCBTYPE=SAM, RECFORM=V and OPEN=EXTEND.

AID standard address interpretation

Indirect addresses, i.e. addresses which precede a pointer operator, are interpreted by default in accordance with the currently valid addressing mode of the debugged object. The %AINT command allows you to deviate from the default address interpretation of AID, i.e. to define whether AID is to work with 24-bit or 31-bit addresses in the case of indirect addressing.

AID standard work area

In conjunction with debugging on machine code level this is the non-privileged area of the virtual memory in a user task, which is occupied by the program and all its connected subsystems.
If no declaration has been made via %BASE and no base qualification has been specified, the AID standard work area applies by default.

AID work area

The AID work area is the address space in which memory locations can be referenced without specifying any qualifications.
In the case of debugging on machine code level, this is the non-privileged part of virtual memory in the user task, which is occupied by the program and all its connected subsystems, or the corresponding area in a memory dump.
Deviation from the AID work area is possible by specifying a base qualification in the address operand of a command. The %BASE command can be used to shift the AID work area from the loaded program to a dump and vice versa.

ALET

Occurs only on ESA systems. The ALET (access list entry token) is a sort of pointer to the access list (AL), via which access to a data space is managed. ALETs are contained in access registers and can be specified in the ALET qualification.

area check

For byte offset and length modification operations and for *receiver* in the %MOVE command, AID checks whether the area limits of the referenced memory objects are exceeded, in which case an error message is issued.

area limits

Each memory object is assigned a particular area, which is defined by the address and length attributes in the case of data names and keywords.
For virtual addresses, the area limits are between V'0' and the last address in virtual memory (V'7FFFFFFF').
The area limits for a CSECT or a COMMON as a memory object are determined by the start and end addresses of the CSECT/COMMON (see chapter 7 in the AID Core Manual [1]).

AR mode

AR (access register) mode is used on ESA systems . It decides whether access registers should be evaluated at the same time as addresses are converted. If %DISPLAY %ASC outputs the hexadecimal value X‘01‘ AR mode is set, access registers are evaluated and addresses can be accessed in data spaces.

attributes

Each memory object has up to six attributes:
address, name (opt), content, length, storage type, output type.
The address, length and storage type can be accessed using selectors. AID uses the name to locate (and analyze) all the associated attributes in the LSD records.
Address constants and constants from the source program have only up to five attributes:
name (opt), value, length, storage type, output type.
They have no address. When a constant is referenced, AID does not access a memory object but merely inserts the value stored for the constant.

base qualification

This is the qualification designating either the loaded program or a memory dump in a dump file. It is specified via E={VM | Dn}.
The base qualification may be globally declared by means of %BASE or specified in the address operand for a single memory reference.

command mode

The term "command mode" in the AID manuals designates the EXPERT mode of the SDF command language. Users who are working in a different mode (`GUIDANCE={MAXIMUM | MEDIUM | MINIMUM | NO}`) should select the EXPERT mode by issuing the command `MODIFY-SDF-OPTIONS GUIDANCE=EXPERT` when they wish to enter AID commands.
AID commands are not supported by SDF syntax, i.e.

– operands cannot be entered via menus and
– AID issues error messages but does not offer a correction dialog.

The system prompt for command input in EXPERT mode is "/".

command sequence

Several commands separated by semicolons (;) form a command sequence, which is processed from left to right. Like a subcommand, a command sequence may contain both AID and BS2000 commands. Certain commands are not permitted in command sequences: this applies to the AID commands %AID, %BASE, %DUMPFILE, %HELP, %OUT, %QUALIFY and the BS2000 commands listed in the appendix of the AID Core Manual [1].

If a command sequence contains one of the commands for runtime control, the command sequence is aborted at that point and the program is started (%CONTINUE, %RESUME, %TRACE) or halted (%STOP). Any subsequent commands in the command sequence are not executed.

constant

A constant represents a value which is not accessible via an address in program memory.
The term "constants" includes the symbolic constants defined in the source program, the results of length selection, length function and address selection, as well as the statement names and source references.
An address constant represents an address. This subset includes statement names, source references, and address selection results. An address constant in a complex memory reference must be followed by a pointer operator (->).

CSECT information

Information contained in the object structure list.

current call hierarchy

The current call hierarchy represents the status of subprogram nesting at the interrupt point. It ranges from the subprogram level at which the program was interrupted, to the hierarchically intermediate subprograms exited by means of CALL statements, to the main program.
The hierarchy is output using the %SDUMP %NEST command.

current CSECT

This is the CSECT in which the program was interrupted. Its name is output in the STOP message.

current program

The current program is the one which is loaded in the task in which the user enters AID commands.

data name

The *dataname* operand stands for all names assigned to data in the source program, i.e. for all variables and constants. Items in structures/tables can be referenced just like in the relevant programming language by means of an identifier or index.

data space

ESA systems have facilities for using other address spaces for data (data spaces), in addition to the program space corresponding to the current address space. Memory objects in a data space can be addressed on machine code level via a virtual address with ALET/SPID qualification.

data type

In accordance with the data type declared in the source program, AID assigns one of the following AID storage types to each data item:

– binary string (≙ %X)
– character (≙ %C)
– numeric (not all data types treated numerically in the relevant programming language correspond to a numeric storage type in AID; see the individual language-specific AID manuals).

Each storage type corresponds to an output type that determines how the data item is output by %DISPLAY or %SDUMP (symbolic debugging).

ESA

(Enterprise **S**ystem **A**rchitecture) is a new hardware concept witth which additional address spaces for data can be utilized. Addressing wthin one of these data spaces is implemented via access registers. When AR mode is activated, the access register corresponding to the basic register is evaluated simultaneously by means of a single command when addresses are converted. Systems with this hardware are called ESA systems; currently these include the H130 and Z5 systems. For the memory concept in ESA systems see the „Executive Macros“ manual.

ESD

The External Symbol Dictionary (ESD) lists the external references of a module. It is generated by the compiler and contains, among other things, information on CSECTs, DSECTs and COMMONs. The linkage editor accesses the ESD when creating the object structure list.

ESV

(External Symbol Vector) is a list of external references for a link and load module (LLM). See also ESD.

external symbol dictionary

If the generation of an external symbol dictionary has not been suppressed, the link editor BINDER will create one on the basis of the ESV (External Symbols Vector).
If the external symbol dictionary was subsequently loaded, you can use %SDUMP %NEST to output the current call hierarchy even if the LSD information was not loaded at the same time

global settings

AID offers commands which serve to adapt the behavior of AID to particular user requirements, save input efforts and facilitate addressing. The global presettings made via these commands are valid throughout the debugging session if not explicitly modified (see %AID, %AINT, %BASE, %OUT and %QUALIFY).

index

An index is part of an address operand and defines the position of an item in a table. It may be specified in the same way as in the programming language or by means of an arithmetic expression from which AID calculates the value of the index.

input buffer

AID has an internal input buffer. If this buffer is not large enough to accommodate a command input, the command is rejected with an error message identifying it as too long. You will then need to abbreviate the command or command sequence or distribute the function over multiple commands.

interrupt point

The address at which a program is interrupted is known as the interrupt point. The STOP message reports the address and the program segment where the interrupt point is located. The program is then continued there. For COBOL85 and FOR1 programs a different continuation address can be specified via %JUMP.

LIFO

Last In First Out principle. If statements from different inputs concur at a test point (%INSERT) or upon occurrence of an event (%ON) the statements entered last are processed first (see chapter 6 in the AID Core Manual [1]).

localization information

Static program nesting for a given memory location is output by AID with
%DISPLAY %HLLOC(memref) for the symbolic level and
%DISPLAY %LOC(memref) for the machine code level.
Conversely, %SDUMP %NEST outputs the dynamic program nesting, i.e. the call hierarchy for the current program interrupt point.

LSD

The List for Symbolic Debugging (LSD) stores the data/statement names defined in the module as well as the compiler-generated source references. The LSD records are created by the compiler and stored in the object module. They are used by AID to retrieve the information required for symbolic addressing.

memory object

A memory object is constituted by a set of contiguous bytes in memory. At the program level, this comprises the program data (provided it has been assigned a memory area) and the instruction code. All registers, the program counter and all other areas which can only be referenced via keywords are likewise memory objects.

Any constants defined in the program, the statement names, source references, results of address selection, length selection and length function and AID literals do not constitute memory objects, however, because they represent a value which cannot be changed.

memory reference

A memory reference addresses a memory object. There are two types of memory reference: simple and complex.

Simple memory references include virtual addresses, a closing C or COM qualification, names for which the address can be obtained by AID from the LSD information, and keywords. Statement names and source references are allowed as memory references in the AID commands %CONTROLn, %DISASSEMBLE, %INSERT, %JUMP, %REMOVE and %TRACE although they are merely address constants.

Complex memory references constitute instructions for AID indicating how to calculate the desired address and which type and length are to apply. The following operations may occur in a complex memory reference: byte offset, indirect addressing, type/length modification and address selection.

monitoring

%CONTROLn, %INSERT and %ON are monitoring commands. When a statement or instruction of the selected group (%CONTROLn) or the defined program address (%INSERT) is encountered in the program sequence or if the selected event occurs (%ON), program execution is interrupted and the specified subcommand is processed by AID.

name range

Comprises all the data and statement names stored for a program segment in the LSD records.

object structure list

On the basis of the External Symbol Dictionary (ESD), the linkage editor TSOSLNK generates the object structure list, provided `SYMTEST=MAP` (the default setting) `SYMTEST=ALL` applies. If the object structure list was subsequently loaded, you can use %SDUMP %NEST to output the current call hierarchy even if the LSD information was not loaded at the same time (see also

external symbol dictionary).
Note, however, that this output will not include details (e.g. source references) for which LSD information is required.

output type

This is an attribute of a memory object and determines how AID outputs the memory contents. Each storage type has its corresponding output type. A list of all
AID-specific storage types together with their output types can be found in the section on “General storage types” in the AID Core Manual [1]. A similar assignment applies to the various data types in different programming languages. A type modification in %DISPLAY and %SDUMP causes the output type to be changed.

pointer operator

This is the string ->, which you enter in an address operand when the contents of a memory object or the value of a constant is used for indirect addressing (see section 7.2.4.2 in the AID Core Manual [1]). The addressing mode is also taken into account for indirect addressing.

program space

In ESA systems, this is the address space in which the program is running. It corresponds to the address space in non-ESA systems.

program state

AID makes a distinction between three program states which the program being tested may assume:

1. The program has stopped.
   The %STOP command, the K2 key, the completion of a %TRACE, or the occurrence of a condition specified with the *control* operand (%INSERT, %ON) has interrupted the program. The task is in command mode.
2. The program is running without tracing.
   The program was loaded and started with START-PROGRAM or started or continued with %RESUME . If no %TRACE has been defined, %CONTINUE can be used for the same purpose.
3. The program is running with tracing.

   %TRACE started or continued the program. The program sequence is logged in accordance with the declarations in the %TRACE command. %CONTINUE has the same effect if a %TRACE is still active.

qualification

A qualification addresses a memory location which is not in the AID work area or not unique therein. The base qualification specifies whether the memory reference is located in the loaded program or in a dump.
Area qualifications specify the path to the program segment containing a memory reference or restrict the effect of a command to the designated area. If an operand qualification is found to be superfluous or contradictory it is ignored. This is the case, for example, if an area qualification is specified for a virtual address.

source reference

A source reference designates an executable statement. It is specified as S'number/name'. *number/name* is generated by the compiler and stored in the LSD records.

SPID

(SPace **ID**entification) is a unique, system-wide identifier for a data space. The SPID is assigned when a data space is created.

statement name

Name assigned to a statement in the source program. This includes the names of labels, entries, paragraphs, sections, etc. An address constant for the name is stored in the LSD records and can be used to address the corresponding memory location.

storage type

This is either the data type defined in the source program or the one selected by way of type modification. AID recognizes the general storage types %X, %C, %P, %D, %F and %A and the special storage types %SX and %S for the interpretation of machine instructions (see %SET and chapter 7 in the AID Core Manual).

subcommand

A subcommand is an operand of the monitoring commands %CONTROLn, %INSERT and %ON. A subcommand consists of a command section which may optionally be preceded by a name and a condition. The command section may consist of a single command or a command sequence and may contain both AID and BS2000 commands. Each subcommand has an execution counter. Information on how an execution condition is formulated, how the names and execution counters are assigned and addressed, and which commands are not permitted within subcommands can be found in chapter 6 of the AID Core Manual [1].

The command section of a subcommand is executed if the monitoring condition (*criterion*, *test-point*, *event*) of the corresponding command is satisfied and any execution condition defined has been met.

tracing

%TRACE is a tracing command. You use it to define the type and number of statements (symbolic debugging) or instructions (machine code level) to be logged.
Program execution is normally traced at the screen, but %OUT %TRACE may be specified to redirect the output to some other output medium.

update dialog

The %AID CHECK=ALL command initiates the update dialog, which takes effect when a %MOVE or %SET is executed. AID queries during the dialog whether updating of the memory contents really is to take place. If N is entered as a response, no modification is carried out; if Y is entered, AID performs the transfer.

user area

Area in virtual memory which is occupied by the loaded program with all its connected subsystems. Corresponds to the area represented by the keyword %CLASS6, %CLASS6ABOVE or %CLASS6BELOW.

Related publications

Ordering manuals

The manuals listed above and the corresponding order numbers can be found in the Siemens Nixdorf *List of Publications*. New publications are described in the *Druckschriften-Neuerscheinungen (New Publications)*.

You can arrange to have both of these sent to you regularly by having your name placed on the appropriate mailing list. Please apply to your local office, where you can also order the manuals.

[1] **AID** (BS2000)
Advanced Interactive Debugger
Core Manual
User Guide

Target group
Programmers in BS2000
Contents
Overview of the AID system
Description of facts and operands which are the same for all programming languages
Messages
Comparison between AID and IDA
Applications
Testing of programs in interactive or batch mode

[2] **AID** (BS2000)
Advanced Interactive Debugger
Debugging of COBOL Programs
User Guide

Target group
COBOL programmers
Contents
Description of the AID commands for symbolic debugging of COBOL programs
Sample application
Applications
Testing of COBOL programs in interactive or batch mode

[3] **AID** (BS2000)
Advanced Interactive Debugger
Debugging of FORTRAN Programs
User Guide

Target group
FORTRAN programmers
Contents
Description of the AID commands for symbolic debugging of FORTRAN programs
Sample application
Applications
Testing of FORTRAN programs in interactive or batch mode

[4] **AID** (BS2000)
Advanced Interactive Debugger
Debugging of PL/I Programs
User Guide

Target group
PL/I programmers
Contents
Description of all the AID commands available for the symbolic debugging of PL/I programs
Sample application

[5] **AID** (BS2000)
Advanced Interactive Debugger
Debugging of ASSEMBH Programs
User Guide

Target group
Assembly language programmers
Contents
Description of the AID commands for symbolic debugging of ASSEMBH-XT programs
Sample application
Applications
Testing of ASSEMBH-XT programs in interactive or batch mode

[6] BS2000
User Commands (SDF Format)
User Guide

Target group
BS2000 users
Contents
BS2000 user commands in SDF (System Dialog Facility) syntax.
Application
BS2000 interactive and batch modes using SDF

[7] BS2000
Executive Macros
User Guide

Target group
BS2000/OSD assembly language programmers
Contents
The manual contains a summary of all Executive macros, detailed descriptions of each macro with notes and examples, including job variable macros, and a comprehensive general training section.

[8] BS2000
Programmiersystem *
(Programming System, Technical Description)

Target group
BS2000 users with an interest in the technical background of their systems (software engineers, system analysts, computer center managers, system administrators).
Computer scientists interested in a concrete example of a general-purpose operating system.
Contents
Functions and principles of implementation of
the linkage editor, static loader and Dynamic Linking Loader
the debugging aids
the program library system
Order number
Only available in German
U3216-J-Z53-1

[9] **ASSEMBH**
Reference Manual

Target group
Users in a BS2000 environment who want to write programs in the assembly or macro language, and to use structured programming.
Contents
Description of the language set of the ASSEMBH assembler in BS2000
Structure of the assembly language; assembler instructions
Structure and elements of the macro language; macro language instructions
Structured programming with ASSEMBH
Predefined macros for structured programming
ILCS interface for structured programming

[10] **ASSEMBH**
User Guide

Target group
Assembly language users under BS2000
Contents
Calling and controlling ASSEMBH
Assembling, linking, loading, and starting programs
Input sources and output of ASSEMBH
Runtime system, structured programming
Language interfacing
Assembler Diagnostic Program ASSDIAG
Advanced Interactive Debugger AID
ASSEMBH messages
Machine instruction formats

[11] **AID** (BS2000)
Advanced Interactive Debugger
Ready Reference

Target group
Programmers in BS2000
Contents
Debugging of programs written in ASSEMBH, C/C++, COBOL, FORTRAN, PL/I and debugging on machine code level
Summary of the AID commands and operands
%SET tables
Comparison of AID and IDA
Applications
Testing of programs in interactive or batch mode

[12] **AID** (BS2000)
Advanced Interactive Debugger
Debugging of C/C++ Programs
User Guide

Target group
C/C++ programmers
Contents
Description of the AID commands for symbolic debugging of C/C++ programs
Sample application
Applications
Testing of C/C++ programs in interactive or batch mode

Index

D

E

Contents

AID V2.1A (BS2000/OSD)

Debugging on Machine Code Level

Target group
Programmers and debuggers
Contents
– Description of the AID commands for debugging on machine code level
– Sample application
– The %SHOW, %SDUMP and %NEST commands are now described, plus context COMMON qualification and (on ESA systems) the ALET/SPID qualifications for data spaces.
  Additional keywords have been included.

Edition: March 1995

File: AID_MC.PDF

Information on this document

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

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

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

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

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

Hinweise zum vorliegenden Dokument

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

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

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

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

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