Stanford Artificial Intelligence Laboratory
Memo AIM-270

October 1975

Computer Science Department
Report No. STAN-B-75-523

BAIL

A debugger for SAIL

by

John F. Reiser

Research sponsored by

Advanced Research Projects Agency
ARPA Order No. 2494
and
National Science Foundation

. Stanford Artificial Intelligence Laboratory
Memo AIM-270

October 1975

Computer Science Department
Report No. STAN-B-75-523

BAIL -- A debugger for SAIL

by

John F. Reiser

ABSTRACT

BAIL is a debugging aid for SAIL programs, where SAIL is an extended dialect of ALGOL60 which runs on the PDP-10 computer. BAIL consists of a breakpoint package and an expression interpreter which allow the user to stop his program at selected points, examine and change the values of variables, and evaluate general SAIL expressions. In addition, BAIL can display text from the source file corresponding to the current location in the program. In may respects BAIL is like DDT or RAID, except that BAIL is oriented towards SAIL and knows about SAIL data types, primitive operations, and procedure implementation.

The work retorted here was funded in part by a National Science Foundation graduate fellowship. Computer facilities provided by Stanford University under the Advanced Research Projects Agency ARPA Contract DAHC15-73-C-0435, and by Institute for Mathematical Studies in the Social Sciences at Stanford

The views and conclusions contained in this document are those of the author(s) and should not be interpreted as necessarily representing the official policies, either expressed or implied, of Stanford University, ARPA, NSF, or the U.S. Government.

Reproduced in the U.S.A. Available from the National Technical Information Service, Springfield, Virginia 22151.

TABLE OF CONTENTS

SECTION 1

INTRODUCTION

The ideal way to debug a computer program is to write it correctly in the first place and not debug it at all. Experience has shown, however, that most programs of moderate size contain errors, and that debugging is a significant part of software product ion. BAIL is a tool which is designed to be useful for interactive debugging of programs written in SAIL [4], a high-level ALGOL-based language for Digital Equipment Corporation (DEC) PDP-10 computers.

In the very early days of computing, debugging was done at the console of the computer. The programmer manipulated switches, observed lights, and had complete control of the whole machine. The programmer could examine and change any location in memory and could start, stop, and single-step the processor. Console debugging soon became uneconomical on medium and large-scale machines. It is still used on minicomputers. This 'type of debugging i s at t ha machine-language level; the lights and switches are direst representations of bits inside the machine.

Debugging moved to the assembly language level with the development of interactive time-sharing systems in the early 1960's. The programmer typed commands at a terminal, and a collection of special subroutines interpreted the commands so that the effect was similar to working at the console of the machine. Instead of communicating in bits, the programmer and subroutines used character strings in the format of octal and decimal integers, text, symbolic machine instructions, and symbolic addresses. One of the most important features of the debugging routines was the ability to suspend the execution of the program being debugged, enter the debugging routines, communicate with the programmer, resume execution, and make the whole process invisible to the program being debugged. This process became known as breakpointing; the location where the main program was stopped is a breakpoint, and the debugging routines are called a breakpoint package. The premier example of a symbolic debugging package is DDT [1], developed for use on the DEC PDP-1 and subsequently extended for use on the PDP-6 and PDP-10. DDT and its derivatives are still among the most powerful tools for debugging assembly language programs.

BAIL is a high-level breakpoint package for use with SAIL programs. (Swinehart [3] and Satterthwaite [2] contain descriptions of other high-level debugging syst ems.) Communication between the programmer and BAIL is in character strings which are the names and values of SAIL objects. BAIL reads general SAIL expressions typed by the programmer, evaluates them in the context of the place in the program where execution was suspended, and prints the resulting value in an appropriate format. The evaluation and printing are performed just as if the programmer had inserted an extra statement into the original program at the point where execution was suspended. BAIL also provides a way to talk about the program, to answer the

questions "Where was execution suspended?", "By what chain of procedure calls did execution proceed to that point?", and "What is the text of the program?"

In order to perform these functions, BAIL must have some information about the program being debugged. The SAIL compiler will produce this information if the program is compiled with an appropriate value supplied for the /B switch. (See the technical portion of the manual for the exact meaning of the various switch values.) In these examples the compiler produces two files. File PROG.REL contains the relocatable code and loader instructions, and file PROGSM 1 contains the information for BAIL. The PROG.SM 1 information consists of the name, type, and accessing information for each variable and procedure, the location of the beginning and end of each statement, and a description of the block structure.

The code for BAIL itself is loaded automatically when the program is loaded. In order for the added Information and code to be of any use, it must be possible to give control to BAIL at the appropriate time. An explicit call to BAIL is possible by declaring EXTERNAL PROCEDURE BAIL; in the program and using the procedure call BAIL;. This works well if it can be predicted in advance where BAILing might be helpful, Runtime errors, such as subscript overflow or CASE index errors, are not as predictable; but responding "B" to the SAIL error handler will activate BAIL. interrupting the program while it is running (to investigate a possible infinite loop, for example) can be achieved under the TENEX operating system by typing control-B. On a DEC TOPS-10 operating system, first return to monitor mode by typing one or more cont rol-C's, then act ivat e BAIL by typing DD<cr>.

BAIL performs some initialization the first time it is entered. The information in the .SM1 file(s) is collected and processed into a file PROG.BAI. This new file reflects ail of the information from the .SM1 files of any separately-compiled programs, and the relocation performed by the loader. If the core image was SAVEd or SSAVEd then in subsequent runs BAIL will use the .BAI file and bypass much of the initialization.

BAIL prompts the programmer for input by typing a number and a colon. The number indicates how many times BAIL has been entered but not yet exited, and thus is the recursion depth inside BAIL. Input to BAIL can be edited using the standard SAIL input-editing characters for the particular operating system under which the program is running. [BAIL requests input via INCHWL on DEC TOPS-10 systems and via INTTY on TENEX systems.] input is terminated whenever the editor act ivat es, string quotation marks balance, and the last character is a semicolon; otherwise input lines are concatenated into one string before being processed further.

The programmer may ask BAIL to evaluate any SAIL expression or procedure call whose evaluation would be legal at the point at which execution of the program being debugged was suspended (except that expressions involving AND, OR, IF-THEN-ELSE, and CASE are not allowed.) BAIL evaluates the expression, prints the resulting value in an appropriate format, and requests further input.

Declared inside BAIL are several procedures whose values or side effects are useful

in the debugging process. These procedures handle the insertion and deletion of breakpoints, display t he static and dynamic scope of the current breakpoint, display selected statements from the source program, allow escape to an assembly-language debugging program, and cause resumption of the suspended main program. These procedures are described in the technical portion of the manual.

The following examples illustrate many of the features available in BAIL. Text was recorded from an actual session on the computer.

SECTION 2

EXAMPLES

This is a test program, run on TENEX.

@*TYPE TEST1.SAI*

```
; <REISER>TEST1.SAI;1   SAT 10-MAY-75 2:37PM          PAGE 1


BEGIN "TEST"
EXTERNAL PROCEDURE BAIL;

INTEGER I, J,K;
STRING A.B.C;
REAL X,Y,Z;
INTEGER ARRAY FOO[0:15]; STRING ARRAY STRARR[1:5,2:6];
INTEGER I TEMVAR DAY; I TEMVAR QQ;

INTEGER PROCEDURE ADD( INTEGER I,J); BEG1 N "ADD"
OUTSTR ("
HI. GLAD YOU STOPPED BY. "); RETURN (I+J) END "ADD";

RECURSIVE INTEGER PROCEDURE FACT (INTEGER N); BEG1 N "FACT"
RETURN (IF N LEQ 1 THEN 1 ELSE N*FACT(N-1)) END "FACT";

SIMPLE PROCEDURE SIMPROC (REFERENCE INTEGER M); BEGIN "SBEG"
ADD (M,M←32) END "SBEG";

FOR I←0 STEP 1 UNTIL 15 DO FOO[I]←I*I;
FOR I←1 STEP 1 UNTIL 5 DO
    FOR J←2 STEP 1 UNTIL 6 DO
        STRARR[I,J]←64+8*I+J;
I←4; J←6; K←112;
A←"BIG DEAL"; B←"QED"; C←"THE LAST PICASSO";

X+3.14153265; Y←0; Z←23.;

BAIL;

ADD(7,45);
SIMPROC (J);

USERERR(0,1,"THIS IS A TEST");

END " TEST";
↑L
```

Compile and load with BAIL.

```
@SAIL.SAV;10
 TENEX SAIL 8.1 4-4-75 (? FOR HELP)
*TEST1,←
**/27B
**
TEST1.SAI;1 1
END OF COMPILATION.
LOADING

LOADER 6+9K CORE
EXECUTION

↑C
```

Save the core image for later use.

```
@SSAVE (PAGES FROM) 0 (TO) 577 (ON) TEST1 [NEW FILE1
 [CONFIRM]
```

Start the program.

```
@START
```

BAIL identifies itself and the files involved.

```
BAIL VER, 1&MAY-75 .
TEST1.SM1;2
  TEST1.SAI;1
End of BAIL initialization.

1: 45;
```

The "1:" is BAIL's prompt. it indicates the level of recursive invocations of BAIL and the fact that BAIL is awaiting input.

See how constants are entered and printed. The "45;<cr>" is typed by the user, and the next line "45" is BAIL's reply.

```
    45
1: 7.089;
     7.083000
1: "SOME RANDOM STRING";
    "SOME RANDOM STRING"
```

An octal constant; all printout is decimal.

```
1:'275;
   189
```

Symbolic constants More than one expression requested

```
1: TRUE,FALSE,NULL;
   -1   0   ""
```

. Variables, assignment

```
1: I;
    4
1: J,X;
    ||    3.141593
1: I←46;
    46
1: I;
    46
```

Relational operators; remember 0 is FALSE.

```
1: I<J;
    0
1: I CEQ J;
    -1
1: 98 LAND '17;
    2
```

An undeclared identifier

```
1: XYZ;

UNKNOWN ID: XYZ
```

Usable as a desk calculator

```
1: 45*(89.4-53.06);
    1635.300
1: X+J;
    9.141593
```

Procedure call

```
1: ADD(3,4);

HI.GLAD YOU STOPPED BY.    7
```

Argument list checking

```
1: ADD(3);

ADD TAKES 2 ARGUMENTS, :ADD(3)
```

Arrays. Array name only gives dimension and subscript bounds information.

```
1: FOO;
    <ARRAY> [0: 151
1: FOO[4];
    16
```

Substring notation has been extended to cover array element s.

```
1: FOO[5 FOR 3];
    25    36    49
1: STRARR;
```

```
    <ARRAY>[ 1:5 2:6]
1:STRARR[1 FOR 2, 4 TO 6];
    "L"   "M"   "N"   "T"   "U"   "V"
```

. Array accesses are interpreted

```
1:FOO[35];

SUBSCRIPTING ERROR.   INDEX    VALUE    MIN    MAX
                        1        35      0      15      : FOO[35]
                                                          ;
```

LENGTH, LOCATION, and MEMORY

```
1:A;
    "BIG DEAL"
1: LENGTH(A);
    8
1:I;
    46
1: LOCATION(I);
    718
1  : MEMORY[718]←64;
     64
1: I;
    64
```

Subst ringing

```
1: A[2 TO INF];
    "IG DEAL"
1:B[3 TO 4];
    "D"
```

Type-in must be terminated by a semicolon

```
1:B
;
    "QED "
```

Tracing of procedure entry and exit

```
1:TRACE("FACT");

1:FACT(4);

 ENTERING FACT 4
  ENTERING FACT 3
   ENTERING FACT 2
    ENTERING FACT 1
    EXITING FACT- 1

   EXITING FACT- 2

  EXITING FACT- 6

 EXITING FACT= 24
   24
1: UNTRACE("FACT");
```

```
1: FACT(5);
   120
```

Breakpoint ing

```
1: BREAK("ADD");
```

```
1: ADD(3,4);
```

Now one level deeper in BAIL recursion. ARGS prints the arguments list.

```
2: ARGS;
   3   4
```

Parameter names evaluate just like variables,

```
2:x;
   3
2:J;
   4
2:K;
   1  1  2
```

To exit from one level of BAIL

```
2:!!GO;
```

```
HI. GLAD YOU STOPPED BY.    7
```

The message is from ADD itself; the value 7 is from BAIL.

Leave another level of BAIL.

```
1:!!GO;
```

And come back again. Where are we?

```
1: TEXT;
```

St at ic block structure

```
LEXICAL SCOPE, TOP DOWN:
$RUN$
TEST
ADD
```

Dynamic procedure invocations. The #4 means coordinate number 4.

```
DYNAMIC SCOPE, MOST RECENT FIRST:
ROUTINE          TEXT
ADD     #4       INTEGER PROCEDURE ADD(INTEGER I,J);BEGI
TEST    #24      ADD(7,45);
SIMPROC (J);

USERERR(0,1,"

1:ARGS;
```

```
   7  45
```

Remove the breakpoint.

```
1:UNBREAK("ADD");

1:!!GO;
```

Output from other calls in the program

```
HI. GLAD YOU STOPPED BY.
HI. GLAD YOU STOPPED BY.
THIS IS A TEST
CALLED FROM 642124 LAST SAIL CALL AT 400303
↑B
```

Entry to BAIL from the error handler

```
1: TEXT;

LEXICAL SCOPE, TOP DOWN:
$RUN$

DYNAMIC SCOPE, MOST RECENT FIRST:
ROUTINE          TEXT
SIMPLE.          '642124 %%% FILE NOT VIEWABLE
TEST     #26     USERERR(0,1,"THIS IS A TEST");

END "T

1: I;

UNKNOWN ID: I
          ;
```

The static scope needs to be set back one on the dynamic chain.

```
1: SETLEX(1);

LEXICAL SCOPE, TOP DOWN:.
$RUN$
TEST

1: I;
   64
1: C;
   " THE LAST PI CASSO"
1: !!GO;

END OF SAIL EXECUTION.
```

Leap and records, DEC TOPS- 10 system.

.TYPE TEST2.SAI

```
BEGIN "TEST"
EXTERNAL PROCEDURE BAIL;
REQUIRE 500 SYSTEM!PDL, 10 PNAMES;

LIST L; SET S,S1,S2,S3,S4,S5;
INTEGER ITEM SUNDAY; ITEM MONDAY,TUESDAY,WEDNESDAY,THURSDAY,FRIDAY,
        SATURDAY;
INTEGER ITEMVAR DAY; ITEMVAR QQ;
ITEMVAR ARRAY P[1:10];

RECORD!CLASS CELL (RECORD!POINTER(CELL) CAR,CDR);
RECORD!POINTER(CELL) CX,CY;

CX←NEW!RECORD(CELL);
CY←NEW!RECORD(CELL);
CELL:CAR[CX]←NULL!RECORD; CELL:CDR[CX]←NULL!RECORD;
CELL:CAR[CY]←CX;CELL:CDR[CY]←NULL!RECORD;

P[1]←SUNDAY; P[2]←MONDAY;
L←{{SUNDAY}};DATUM(SUNDAY)←0;DAY←SUNDAY; Q&MONDAY; S←{QQ};
S1←{SUNDAY,MONDAY,TUESDAY,WEDNESDAY,THURSDAY,FRIDAY,SATURDAY};
S2←{MONDAY,TUESDAY,WEDNESDAY,THURSDAY,FRIDAY};
S3←{MONDAY,WEDNESDAY,FRIDAY};   S4←{SATURDAY,SUNDAY};
S5←{SUNDAY,FRIDAY};

FOREACH DAY SUCH THAT DAY IN S1 DO MAKE DAY XOR SUNDAY EQV SATURDAY;

BAIL:

USERERR(0,1,"THIS IS A TEST");

END "TEST":
```

```
EXIT
↑C
.EXECUTE TEST2.SAI(27B,)
SAIL: TEST2 1
LOADING
LOADER 15K CORE.
25K MAX 153, WORDS FREE
EXECUTION

BAIL VER. 10-MAY-75
TEST2. SM1
  TEST2.SAI
END OF BAIL INITIALIZATION.

1: L;
   {{SUNDAY}}
```

```
1: S4;
    {SUNDAY,  SATURDAY)
1: S5;
    (SUNDAY, FRIDAY)
1: S4 UNION S5;
    {SUNDAY, FRIDAY, SATURDAY)
1: FRIDAY IN S4;
    0
1: S2 LEQ S2;
    -1
1: DAY;
    SATURDAY
1: DATUM(DAY);
    0
1: cx;
    CELL.9231
1: CELL:CAR[CX];
    NULL ! RECORD
1: CELL:CAR[ CY ];
    CELL.9231
1: SUNDAY ASSOC SATURDAY;
    {SUNDAY}
1 : SUNDAY EQV SATURDAY;
    (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY)
1: SUNDAY XOR SATURDAY;
    PHI
1: SUNDAY EQV SUNDAY;
    PHI
1: ↑C
```

Go back to the earlier example

```
@TEST1.SAV;1
```

Initialization uses file created last time.

```
BAIL ver. 10-May-75 using TEST1.BAI
End of BAIL initialization.
```

Switch /27B at compile-time makes SAIL predeclared runtime routines known to BAIL,

```
1: OPENFILE(NULL,"W");
TODAY.TMP
   4
1: OUT(4,"TH IS IS A TEMPORARY FILE CREATED WHILE IN BAIL.");

1: CFILE(4);
   -1
1: OPENFILE("","RC");
TODAY.TMP [OLD VERSION]
   4
1: SINI(4,200,"Z");
   "THIS IS A TEMPORARY F IL E CREATED WHILE I N BAIL."
1 : ODTIM(-1,-1);
   "SATURDAY, HAY 10, 1975 17:19:29"
```

Quickie review of BAIL capabilities

```
1:?

EXPRESS ION;
PROCEDURE!CALL;
TRACE("PROCEDURE");
UNTRACE("PROCEDURE");
BREAK("PROCEDURE, BLOCK, OR LABEL");
UNBREAK("PROCEDURE, BLOCK, OR LABEL");
!!GO;
SETLEX(LEVEL);
TEXT:
ARGS;
HELP:
DDT;
?

1:↑C
```

End of the examples.

SECTION 3

COMPILE-TIME ACTION

The principal result of activating BAIL at compile-time is the generation of a file of information about the source program for use by the run-time interpreter, This file has the same name as the .REL file produced by the compilation, except that the extension is .SM 1. If requested, BAIL will also generate some additional code for SIMPLE procedures to make them more palatable to the run-time interpreter.

The action of BAIL at compile time is governed by the value of the /B switch passed to the compiler. If the value of this switch is zero (the default if no value is specified) then BAIL is completely inactive. Otherwise, the low-order bits determine the actions which BAIL performs. [The value of the /B switch is interpreted as octal.]

bit action

1 If this bit is on, then the .SM1 file will contain the program counter to source/listing text directory.

2 If this bit is on, then the .SM1 file will contain symbol information for ail SAIL symbols encountered in the source. If this bit is off, then information is kept only for procedures, parameters, blocks, and internals; i.e., non- internal local variables are not recorded.

4 If this bit is on, then SIMPLE procedures will get procedure descriptors, and one additional instruction (a JFCL 0, which is the fastest machine no-op instruction) is inserted at the beginning of SIMPLE procedures. Except for these two changes, all properties of SIMPLE procedures remain the same as before. The procedure descriptor is necessary if the procedure, is to be called interpretively or if the procedure is to be TRACEd.

'10 If this bit is on, then BAIL will not-be automatically loaded and initialized, although all other actions requested are performed. This is primarily intended to make it easier to debug new versions of BAIL without interfering with SYS:BAIL.REL. By using this switch the decision to load BAIL is delayed until load time.

'20 If this bit is on, then a request to load SYS:BAIPDn.REL is generated. This file contains procedure descriptors for most of the SAIL predeclared runtime routines, making it possible to call them from BAIL. The procedure descriptors and their symbols occupy about 6K.

The B switch must occur on the binary term, not the listing or source term. Thus:

```
.R SAIL                    or    ● OM PROG(27B,)
*PROG/27B←PROG
```

The program counter to source/listing index is kept in terms of coordinates. The coordinate counter is zeroed at the beginning of the compilation and is incremented by one for each BEGIN, ELSE, and semicolon seen by the parser, provided at least one word of code has been compiled since the previous coordinate was defined. Note that COMMENTs are seen only by the scanner, not the parser, and that DEFINEs and many declarations merely define symbols and do not cause instructions to be generated. For each coordinate the directory contains the coordinate number, the value of the program counter, and a file pointer to the appropriate place. The appropriate place is the source file unless a listing file is being produced and the CREF switch is off, in which case it is the listing file. [The listing file produced for CREF is nearly unreadable.], On a non-CREF listing, the program counter is replaced by the coordinate number if bit 1 . of the /B switch is on.

The symbol table informat ion consists of the block structure and the name, access information; and type for each symbol.

If a BEGIN-END pair has declarations (i.e., is a true block and not just a compound statement) but does not have a name, then BAIL will invent one. The name is of the form Bnnnn where nnnn is the decimal value of the current coordinate.

SECTION 4

RUN-TIME ACTION

The BAIL run-time interpreter is itself a SAIL program which resides on the system disk area, This program is usually loaded automatically, and does some initialization when entered for the first time. The initialization generates a .BAI file of information collected from the .SM1 files produced by separate compilations (if any). The .SM1 files correspond to .REL files, and the .BAI file corresponds to the .DMP or .SAV file, Like RPG or CCL, BAIL will, try to bypass much of the initialization and use an existing ● AI file if appropriate. During initialization BAIL displays the names of the .SM1 files it is processing. For each .SM1 file which contains program counter/text index information, BAIL displays the names of the text files and determines whether the text files are accessible.

The interpreter is activated by explicit call, previously inserted breakpoints, or the SAIL error handler. For an explicit call, say EXTERNAL PROCEDURE BAIL; . . . BAIL;. From the error handler, respond B. Breakpoints will be described later in this section.

4.1 - Debugging Requests

When entered, BAIL prints the debugging recursion level followed by a colon, and awaits a debugging request. BAIL accepts ALGOL and LEAP expressions of the SAIL language. A complete description is given in [4] and in the addenda describing the syntax of records and record-pointers. The following exceptions should be noted. Expressions involving control structure are not allowed, hence BAIL will not recognize AND, CR, IF-THEN-ELSE, or CASE. Bracketed triple items are not allowed. The TO and FOR substring and sublist operators have been extended to operate as array subscript ranges, FOR PRINT-OUT ONLY. If FOO is an array, then FOO[3 TO 7]; will act like FOO[3], FOO[4], FOO[5], FOO[6], FOO[7]; but is easier to type. This extension is for print-out only; no general APL syntax or semantics are provided.

BAIL evaluates symbolic names according to the scope rules of ALGOL, extended to always recognize names which are globally unique and have a fixed memory location (everything except parameters and recursive locals). For any activation of BAIL, the initial scope is the ALGOL scope of the statement from which BAIL was activated. The procedure SETLEX (see below) may be used to change the scope to that of any one of the links in the dynamic activation chain.

Several procedures are predeclared in the outermost block to handle breakpoints and display information. These are described individually below.

4.16 - ARGS

STRING PROCEDURE ARGS;

The arguments to the procedure which was most recently called,

4.3 - BREAK

PROCEDURE BREAK("location","condition"(NULL),"action"(NULL),count(0));

BREAK inserts a breakpoint. The syntax for the first argument is

<location>::=<label>|<procedure>|<block name>|#<nnnn>
|<block name><delim><location>
<delim>::=<any character not legal in an identifier>
<nnnn>::=<decimal coordinate number>

If the location is specified by the <block name><delim><location> construct then the blocks of the core image are searched in ascending order of address of BEGINs until the first <block name> is matched. The search continues until the second <block name> is matched, etc. The breakpoint is inserted at the label, procedure, or coordinate declared within the scope of the last <block name>. This detailed specification is not usually necessary, as shown in the examples. The last three parameters are default able and need not be specified, again as in the examples. The action taken at a breakpoint is

```
IF LENGTH( condition) AND EVAL( condition) AND (count ←count - 1)<0 AND
    LENGTH( action) THEN EVAL( action);
EVAL( TTY);  .
```

Here EVAL is a procedure which evaluates its string argument and returns the value of the last expression evaluated (similar to PROGN in LISP).

4.4 - DDT

PROCEDURE DDT; .

This procedure transfers control to an assembly language debugging program (if one was loaded). ,

4.5 - HELP

PROCEDURE HELP;

A list of options, including short descriptions of the procedures described in this section, is printed. A question mark followed by a carriage return is interpreted as a call to HELP.

4.6 - SETLEX

PROCEDURE SETLEX(level);

Evaluating SETLEX(n) changes the static (lexical) scope to the scope of the n-th entry in the dynamic scope list. SETLEX(0) is the scope of the breakpoint; SETLEX(1) is the scope of the most recent procedure call in the dynamic scope, etc.

4.7 - SHOW

STRING PROCEDURE SHOW(first, last(0));

The text of the program from the source or listing file. If last is less than first then set last to last+first. Return coordinates first through last. SHOW(5,3) gives coordinates 5, 6, 7, and 8; SHOW(5,7) gives coordinates 5, 6, and 7; SHOW(5) gives coordinate 5 only.

A plus sign ("+") following the coordinate number indicates that the values of some variables have been carried over in accumulators from the previous coordinate, Changing the value of variables might not be successful in such a case, because BAIL will not change any accumulator . value directly. The MEMORY construct can be used to modify any location in a core image, including the accumulators.

4.8 - TEXT

STRING PROCEDURE TEXT;

The current static and dynamic scopes, with text from the source or listing file.

4.9 - TRACE

PROCEDURE TRACE("procedure");

Special breakpoints are inserted at the beginning and end of the procedure named. On entry, the procedure name and arguments are typed. On exit, the name and value returned (if any) are typed.

4.10 - TRAPS

STRING PROCEDURE TRAPS;

A list of the current breakpoints and. traces. ,

4.11 - UNBREAK

PROCEDURE UNBREAK("locat ion");

The breakpoint at the location specified is removed.

4.12 - UNTRACE

PROCEDURE UNTRACE("procedure");

The breakpoints inserted by TRACE are removed.

4.13 - GO

pseudoPROCEDURE !!GO;

An immediate exit from the current instantiation of BAIL is taken and execution of the program is resumed. !!GO is a reserved word (the only one) in BAIL.

4.14 - GSTEP

pseudoPROCEDURE !!GSTEP;

Temporary breakpoints are inserted at all of the logical exits of the current statement, and execution of the program is resumed. Logical exits are the next statement and locations to which the current statement can jump, excluding any procedure calls. All of the breakpoints which are inserted will be removed as soon as one of them is encountered.

4.15 - STEP

pseudoPROCEDURE !!STEP;

Temporary breakpoints are inserted at all locations. to which the current statement can jump, including procedure calls, and execution of the program is resumed.

4.16 - STRING TYPEOUT

Strings are usually typed so that the output looks the same as the input, i.e., a string is typed with surrounding quotation marks and doubled internal quotation marks. For SHOW, ARGS, and TEXT this would ordinarily create confusion, so they are handled specially. When these procedures are evaluated they set a flag which inhibits quotation mark fiddling, provided that no further evaluation takes place before the next typeout. Thus SHOW(5,3); will be typed plain, but STR←SHOW(5,3); will have quotation marks massaged.

4.17 - BAIL and DDT

If BAIL is initialized in a core image which does not have DDT or RAID, then things will be set up so that the monitor command DDT gets you into BAIL in the right way. That is, BAIL will be your DDT. To enter BAIL from DDT (provided that the SAIL initialization sequence has already been performed), use

```
pushi P,<program counter>$X
JRST BAIL$X
```

For example, if .JBOPC contains the program counter,

```
PUSH P,.JBOPCSX
```

```
JRST    BAILSX
```

The entry B. provides a path from DDT to BAIL which works whether or not the core image has been initialized. One use of this feature is to BREAK a procedure in an existing product ion program without recompiling. For example,

```
@; PROG originally compiled, loaded with BAIL and DDT, and' SAVEd
@GET PROG
@DD
 B.$G
BAIL  initialization
        :
        :
1:BREAK( "procedure");
1:!!GO;

$G
```

To enter DDT from BAIL, simply say DDT;. For operation under TENEX, control-B is a pseudo-interrupt character which gets you into BAIL.

4.18 - WARNINGS

Since BAIL is itself a SAIL procedure, entering BAIL from the error handier or DDT after a push-down overflow or a string garbage collection error will get you into trouble.

SIMPLE procedures cause headaches for BAIL because they do not keep a display pointer. [Indeed, the compiler gets lost in the following example, and does not complain:

```
BEGIN "LOST"
PROCEDURE A(INTEGER I); BEGIN "A"
  SIMPLE PROCEDURE B; OUTSTR("THE VALUE OF I IS " & CVS( I));
  PROCEDURE C(INTEGER J); B;
C(2);
END "A";

A(1);
END "LOST";
```

BAIL tries valiantly to do the right thing, but occasionally it also gets lost. BAIL will try to warn you if it can. In general, looking at value string parameters of SIMPLE procedures does not work.

SECTION 5

RESOURCES USED

I. Compile-time

A. One channel. This means that REQUIREd source files may only be nested to a depth of about 9.

B. Memory. Up to 1 1*(maximum lexical nesting 'depth) more words of memory may be required compared with previous- compilations.

C, CPU time. Approximately 0.3 seconds per. page of dense text.

II. Run-t ime

A. Channels. Three during, initialization, two thereafter. Channels are obtained via GETCHAN.

B. BAIL uses 7 of the privileged' breaktables, obtaining them via GETBREAK.

C. REQUIRE 64 STRING!PDL. Necessary if the debugging recursion level will exceed 3 or 4.

D. Memory. (9.5K +((# of coordinates+127) DIV 128) + (2* # of blocks) + (5* # of symbols)) words.

E. CPU time.

1, Initialization. Typically 4 seconds for a 30 page program.

2. Debugging requests. 0.07 seconds per simple request. DDT response time.

III. Disk space

A. The .SM1 file for a /7B compilation is typically one-fourth the size of the corresponding .REL file.

B. The .BAI file for a group of /7B compilations is typically one-third the total size of the corresponding .REL files.

SECTION 6

CURRENT STATUS

The state of the world is determined by the values of the accumulators and the value of the SAIL variable !SKIP!.

The run-time interpreter recognizes only the first 15 characters of identifier names; the rest are discarded without comment. The characters which are legal in identifiers are

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
0123456789!_αβπλc⊃∀∃→~#$\|
```

Notable for its absence: period.

LOCATION of a procedure does not work.

PROPS is read-only.

Bracketed triple items are not allowed.

A procedure call containing the name of a parametric procedure (functional argument} is not handled properly.

Contexts are not recognized.

The run-time interpreter will not recognize macros.

External linkage: If an identifier is never referenced by code (i.e., has an empty fixup chain at the time fixups are put out to the loader) then that identifier is not defined by SAIL. Thus variables which are never used do not take up space, and a request to the loader is not -made for EXTERNALS which are not referenced. This feature of SAIL is cast in concrete and will not be changed. As a result, the following DOES NOT WORK unless special precautions are taken:.

```
BEGIN
EXTERNAL PROCEDURE BAIL;
EXTERNAL PROCEDURE PLOT(REAL X0,Y0,X1,Y1);
REQUIRE "CALCOM" LIBRARY;

BAIL END
```

PLOT will not be defined by SAIL, hence BAIL will not know about it. However if there

are any references to PLOT (real or "dummy" calls) then BAIL will know. The following trick can also be used, assuming that CALCOM is a SAIL-compiled procedure: Compile CALCOM with / 108, which says "make the .SM 1 file but don't automatically load SYS:BAIL.REL". Then the above will win (due to BAIL recognizing things which are globally unique) and programs which do not use BAIL will not have it loaded just beacuse the library was used. This same problem occurs with EXTERNAL RECORD!CLASS declarations, Use of the subfield index information does not cause a reference to the class name but NEW!RECORD does. Thus the same /10B trick must be used if there are no NEW!RECORD calls.

REFERENCES

[1] DECsystem10 Assembly Language Handbook DEC-10-NRZC-D, Digital Equipment Corporation, Maynard, Massachusetts, 1973.

[2] Edwin H. Satterthwaite Jr., "Source Language Debugging Tools" (Ph.D. thesis), Computer Science Department, St anford University, May 1975.

[3] Daniel C. Swinehart, "COPILOT: A Multiple Process Approach to Interactive Programming Systems" (Ph.D. thesis), Computer Science Department, Stanford University, August 1974.

[4] Kurt VanLehn (ed.), SAIL USER MANUAL, Stanford Artificial Intelligence Laboratory memo AIM-204 (Computer Science Department report STAN-CS-73-373), July 1973.