Swat, a BCPL-oriented debugger

Swat is a debugger meant to be used with the ALTO operating system. While many of its features are BCPL oriented, it can be used on any NOVA code program. This document describes version 18 of Swat.

1. Invocation

Swat may be applied to any program running under the operating system after it has been installed (see Installation below). There are four ways of getting its attention:

(1) Press the lowest, right-hand key of the ALTO keyboard, together with the <control> and <shift> keys at the left.

(2) Have your program execute the op-code 77400B.

(3) Invoke the Resume/S command (see below).

(4) Boot the file Dumper.Boot, normally by booting with the "DU" keys depressed.

(5) Type <programname>/! to the Alto command processor.

(6) Call the function CallSwat. Up to 2 arguments will be printed as BCPL strings. Thus CallSwat("No more memory")

2. Commands

The command has suffix action symbols, all control characters (e.g. ↑C). "n" is any BCPL expression (see Expressions below), "$" is escape except where noted, "cr" means carriage return, "lf" means line-feed.

2.1. Displaying cells

n↑D prints the contents of n in decimal
n↑I prints the contents of n as two 8-bit bytes
n↑N prints the contents of n as a NOVA instruction
n↑O prints the contents of n in octal
n↑S prints the contents of n as a pair of characters
n↑V prints n (in octal) rather than its contents

The last cell printed is called the open cell. ↑O, ↑D, ↑I, ↑N, or ↑S alone re-prints the open cell in the appropriate format. If you wish to print out a number of cells, beginning with the open cell, say n$↑D, n$↑I, etc. This command will not change the identity of the open cell.

If (↑J) prints the contents of the next cell (after the open one) in the same mode.

↑L

↑W prints the cell before the open cell.

↑A prints the cell pointed at by the open cell.

↑E treats the open call as a NOVA instruction, computes its effective address and prints its contents.

2.2. Changing cells

The contents of the open cell (if there is one) may be changed by typing an expression for the new value followed by a cr or lf.

2.3. Running the program

↑P resumes the program, i.e. proceeds.

n↑G resumes the program at n, i.e. goes there.

<e >$<e >$...$<e >↑C calls procedure <e > with parameters <e >,...,<e > (n<6). If you wish one of the arguments to be a BCPL-format string, merely enclose it in quotes. Thus OpenFile$"Com.Cm."↑C will return a stream on the file.

↑U restores the user's screen. Hitting the break key brings back Swat.

↑K forces the user programs to abort.

2.4. Break Points

n↑B sets a break at n

↑B set a break at the open cell

0$n↑B deletes the break at n

0$$↑B deletes all breaks

$$↑B prints all broken locations and checks that they haven't been clobbered.

$↑P removes the current break and proceeds.

n$↑P sets a break at a BCPL return point in the stack somewhere and proceeds from the present break. The parameter n specifies the frame. Thus if ↑T typed out 0:GOO+56 1:HAM+5, 1$↑P would set a break at HAM+6 and proceed.

2.5. Stack Study

↑T prints the current PC and all return addresses in the call stack (symbolically), until an inconsistency is found in the stack. After each return address is listed the parameters

↑L

passed to the procedure that will be returned to. Thus, if you see an entry like "3: FindIt+45--(14 177777)", the procedure FindIt was called with arguments 14b and -1.

n↑T prints n (or less) return addresses.

n↑F prints the parameters of the nth latest stack frame and sets the pseudo symbol "$" (not escape) equal to the base of that frame. If ↑T displayed something like 0:FOO+3, 1:BLETCH+10,... Type 1↑F to see the parameters that were passed to BLETCH. $ is set to the base of BLETCH's frame.

2.6. Symbol table

↑Y prompts to get a symbol file. Type the name of the subsystem that's running. (If BLDR created the file FOO it also created FOO.SYMS which gives the locations of all the static names. Only statics can be used in Swat.)

2.7. Save/Restore

↑Q saves the current SWATEE on a file (prompts) (see below)

↑L makes a (prompted for) file the current SWATEE (see below)

2.8. The Spy Facility

The spy can be used to estimate where the time is going on a percentage basis. (It samples the PC every 30-milliseconds).

(1) Type ↑X and Swat will display how much user memory it needs for the metering code and tables.

(2) Probe around to find a block of storage of the required size, and tell Swat by typing

n↑X

where n is the first word of the block.

(3) Proceed to run the program.

(4) Once Swat gets control again you can type

$↑X

to display the results and terminate the spying activity, or

$$↑X

to display the results so far and continue the spying.

↑L

2.9. Miscellaneous

$↑Y Will prompt for the name of a (text) file from which Swat commands should be taken. Reading will continue across "proceeds" from breakpoints, but will be aborted if Swat is invoked by the keyboard or by the standard break-point trap (77400b).

n↑R Prints out the value of an R or S register n. You must have a RAM for this to work, and n cannot be 37b or 77b.

2.10. Examples

X↑O↑D prints the value of X in octal, then decimal.

FUNC+3+↑N !f !f prints instructions 3, 4, and 5 of FUNC.

1↑O7 sets location 1 to 7.

LABEL↑B sets a break at LABEL

7562↑B sets a break at location 7562B

SQRT$16↑C calls the (user) function SQRT (the returned value is printed)

LABEL+3↑G transfers to the third instruction after LABEL.

0↑T prints the PC

0↑F prints the parameters of the most recent call

2↑F prints the parameters of the third most recently called procedure; then

$↑O prints the saved stack pointer (FLAST)

$+1↑O prints the return address (FRET)

$+6↑O prints the first local (if the procedure has 2 parameters).

3. Expressions

Expressions are as in BCPL with the following exceptions

'	means exclusive or
\	means remainder
|	means lshift for positive arguments, rshift for negative
	means NOT

A string of digits is interpreted as octal unless suffixed by a "."

$ (not escape) is the base of the last opened stack frame (see ↑F above). Initially it is the last frame.

. is the last opened cell

.PC is the address (sic) of the PC

.AC1,...,.AC3 are the addresses of the accumulators

.CRY is the address of the carry bit

No function calls in expressions.

No relational operators (e.g. EQ)

No conditional expressions

No lv operation

3.1. Examples

.-1↑O prints the cell before the currently open cell.

.+1↑O is like line-feed.

.AC1↑O6 sets AC1 to 6

.PC↑O72
↑P is like 72↑G

.PC↑O lf lf lf lf prints the PC and the AC's

The conventions for expression evaluation are not truly BCPL-like. "F↑O" will print the first instruction of F if BLDR thought it was a procedure or label, but print the contents of static cell F if BLDR thought it was a variable. If F started life as a variable, but had a procedure assigned to it you must call it by "@F↑C" instead of "F↑C".

4. Resumable Files

The file SWATEE is a snapshot of a running program and can be saved for subseqent resumption or examination. You can create a copy of SWATEE by using COPY or, if you are in Swat, typing ↑L and giving a file name. This copies SWATEE to the named file and appends some information internal to Swat -- the current symbol table and break point data.

There are several ways to restart resumable files:

1. Press the boot button while holding down the keys for the file.

2. Type the command (it is interpreted by the command processor)

RESUME file

If "file" is omitted SWATEE is assumed.

RESUME/S file
↑L

writes file onto SWATEE and invokes Swat.

3. While in Swat, type ↑Q and give a file name. The file is copied onto SWATEE and Swat's internal information is restored to whatever was saved by the ↑L command that created the file. If the file was created in some way other than ↑L, the internal information is reset to an empty state.

5. Invoking Swat with the Boot Button

At any time, press the boot button while holding down the keys for the file Dumper.Boot (hopefully "DU"). This writes the existing memory onto SWATEE with the omission of page 0 which is lost. Also the display word (420) is cleared. Finally, Swat is invoked.

6. Error Message Printing

Swat contains some facilities to aid in printing error messages. Because the Swat resident is almost always present when a program is running, an error message can be printed by simulating a Swat "break," and letting the Swat program decipher the error specification and print a reasonable message.

If Swat is invoked by the #77403 trap instruction, the contents of AC0 are taken to be a pointer to a BCPL string for a file name; AC1 is a pointer to table [errCode%ClearBit; p1; p2; p3; p4....], where errCode (0 le errCode le 32000.) is an error code, the p's are "parameters," and ClearBit is either #100000 (clear the Swat screen before printing the message) or 0 (do not clear).

The intended use is with a BCPL procedure like:

```
let BravoError(code, p1, p2, nil, nil, nil) be
   [
   code=code%UserClearScreenBit
        (table [ #77403; #1401 ] )("bravo.errors", lv code)
   // do a "finish" here if fatal error
   ]
```

The error messages file is a sequence of error messages, searched in a dumb fashion. An error message is:

```
a. An unsigned decimal error number (digits only)
b. Followed optionally by:
        C       Always clear the screen before printing the message
        M       (see below)
        L       Log the error via the Ethernet.
c. Followed by a <space>.
d. Followed by text for the message, including carriage returns, etc.
If you wish to refer to a parameter, give:
        $
        followed by a digit to specify the parameter number (1,2,....)
        followed by a character to say how to print the parameter:
                O = octal
                D = decimal
↑L
```

S = string (parameter is pointer to BCPL string)
(example: $1D will print parameter 1 in decimal)
e. Followed by $$.

After the message is typed, if M was specified, the message "Type <control>K to kill, or <control>P to proceed." is typed out.

7. Parity Error Information

When the Alto detects a parity error, Swat is usually invoked to print a message about the details of the error. It then attempts to "log" the error with an Ethernet server responsible for keeping maintenance information. If the server is not operating, or if your Alto is not connected to an Ethernet with such a server, simply strike the "Swat" key (<blank-bottom>), and the familiar "#" will appear.

In many cases, you will want to continue execution of your program after a parity error is detected. Simply type <control>P to Swat.

8. Installation

Get the file InstallSwat.Run. Then invoke it to will create SWAT (the debugger), SWATEE (the swapout file for the user's memory image), and Dumper.Boot. InstallSwat.Run may be deleted after it has been run once. Use BOOTKEYS to discover the keys to depress for Dumper.Boot. If the answer is not "DU" invoke

MOVETOKEYS Dumper.Boot DU

9. Caveats

1. Swat has about 1k of resident code in high core. This code is not changed when new subsystems come in. Therefore re-boot if it seems to be in a bad state.

2. Instruction 77400B is used for breaks, and location 567B (in the trap vector) is used.

3. Interrupt channel 8 (00400B) is used for keyboard interrupts.

4. The resident disables interrupts on entry and enables them on exit (clobbering 500B) so putting breaks in non-interruptable code is dubious.

5. A program fetching data from a broken location will get 77400B.

6. While most interrupt routines are reasonably polite and always resume the interrupted code where it left off, the politeness of Swat's keyboard interrupt is entirely in the hands of the person at the ↑L

controls. If he re-starts by saying ↑P, all goes well; but he may say ↑G or ↑C. Therefore

(1) You should disable the keyboard interrupt by anding 77377B into 453B during critical sections of code (once they are debugged).

(2) Expect occasional anomalies after ↑C or ↑G is used.

7. The mappings between symbols and addresses are naive about BCPL's block structure

a) If a symbol is defined twice or more you get the lowest address.

b) An address is mapped into a procedure name plus a displacement for symbolic type out (e.g. for ↑T). If procedure A is defined inside procedure B, most of B's addresses will be typed as if they were A's.

8. If a disk error prevents swapping the offending disk control block and label are displayed in the "boot-lights" manner.

9. Locations 700 through 706 are used to save the registers before each swap.

10. If a file created on a different disk is resumed by booting, invoking Swat may not work because SWAT and SWATEE may not reside at the same disk addresses on the different disks. This difficulty does not occur if the RESUME command is used.
↑L