## Identification

Published: 6/7/66

Breakpoint processor
breaker
D. B. Wagner

## Purpose

Breaker accepts requests to interrupt the execution of a program upon the occurrence of certain events. The tracer command (see BX.10.02) is normally used to specify actions to be performed at each break.

## Usage

The command

```
breaker
```

causes breaker to begin reading requests from the console. The user may type any of the requests listed below or any of the "control" requests (if, else, do, end) described in BX.10.00. He may also type macro invocations (in the same form as in the command language: see BX.1.01) which expand to sequences of these requests. If a line received by breaker (after macro expansion) is not recognizable as a request, it is treated as a command. The line is given to the Shell, which gives an appropriate diagnostic if it is not a command either. *

## Requests to Breaker

The request

```
setbreak name event
```

causes arrangements to be made with the System so that when the specified event (see below) occurs breaker will regain control and make a call to the tracer entry tracer$report, then allow the program to resume running. Name is a character-string expression (see the discussion of expressions in BX.10.00) which is to be used as the first (identification) argument in these calls. Event is one of the following (meanings are usually clear: detailed explanations are avoided here to keep this document to a manageable size)

```
extime n ms
extime n sec
extime n min
extime n hrs

realtime (same arguments as extime)
```

access *spec* *location* to *location*
access *spec* *variable*
(spec= a combination of X (execution), R (read), W (write))

call from *seg*
call from *seg$expression*
call to *seg*
call to *seg$expression*
call from ... to ...

return (same arguments as *call*)

extref (same arguments as *call*)

*Call* and *return* refer to subroutine calls and returns (as in the CTSS command STRACE). These breaks and the *extref* (external reference) break are implemented using special entries to the Linker similar to those described in BE.12.01 for the 645 simulator system.

The *access* event break is implemented by temporarily changing the descriptor bits for the segment involved and arranging to have access-violations reflected to the debugger's trap-handling routines. This method can of course be rather costly, especially if a small block out of a large segment is specified, since all accesses of the specified type anywhere in the segment must be executed interpretively. It is expected that interpretive execution of instructions will take an average of 50x normal execution time.

There is a problem with the *extime* (execution time) event: within the system execution times are measured in core cycles, not in conventional time units. Hence in addition to the time units mentioned above for the *extime* request (which have to be approximated in terms of core cycles), the units *kc*, *mc*, and *gc* (kilocycles, megacycles, and gigacycles) are provided.

The request

*exit*

causes *breaker* to return to its caller, normally the Shell.

*Examples*

A "watch" macro might be defined to permit the monitoring of the values of variables through time. When invoked to watch the variable *beta* in a PL/I program and report every 10 ms., the macro might expand to the following sequence, which as can be seen includes both commands and requests:

```
breaker                          (command)
setbreak "xyz" extime 10 ms      (request)
exit                                "
tracer                           (command)
setaction "xyz"                  (request)
probe                            (command to be stored)
print "beta" beta                (request to be stored)
proceed                             "
endaction                        (request)
exit                                "
```

When the program is started up, the following lines, interspersed of course with normal program output, might be typed on the console:

```
beta          6.723
beta          8.927
beta          5.400
beta          6.723
beta          8.927
```

and this might or might not give the user a clue to what is going wrong with his program.

A macro to perform the same function as the "B" request in FAPDBG (break when control passes to a specified location and begin reading requests) might expand to:

```
breaker                          (command)
setbreak "xyz" access X location
                                 (request)
exit                                "
tracer                           (command)
setaction "xyz"                  (request)
probe                            (command to be stored)
print "BREAK"                    (request to be stored)
endaction                        (request)
exit                                "
```

When and if control reaches the location specified, the command probe will be called. It will print "BREAK" and begin reading requests. When the user eventually types the proceed request, the program will start running again.