[7.7] Use *return* instead of *stop* in programs for better flexibility, and to avoid a crash

Note: this bug has been fixed in AMS 2.05. stop now works correctly in archived programs

The *stop* instruction stops program execution. The *return* instruction, used without an argument, also effectively stops program execution. It is better to use *return* instead of *stop* for two reasons. First, it makes your programs more flexible. Second, under certain circumstances, *stop* can cause a crash that can be fixed only by resetting the calculator.

The improved flexibility comes about if you call one program from another program. For example, suppose you wrote a program called *app1()* which does something useful. Later, you decide it would be helpful to call *app1()* from another program. If you use *stop* in *app1(),* execution stops when *app1()* is finished, when you really want to return to the calling application.

Further, using *stop* instead of *return* can cause a calculator crash. The effect of this bug is to lock up the TI92 so that it does not handle keypresses and must be reset. I have only verified this bug on my TI92 with the Plus module installed, ROM version 1.05.

The bug occurs when an archived program containing the *stop* instruction is executed with the *expr()* instruction. To duplicate the bug, create this program in the \main folder:

```
stopit()
prgm
stop
endprgm
```

Archive *stopit(),* then, at the command line, enter

```
expr("stopit()") [ENTER]
```

The BUSY annunciator in the LCD turns on and does not turn off. Keys seem to be recognized, but not handled. For example, pressing [green diamond] will toggle the 'diamond' annunciator, and pressing [2nd] will toggle the '2nd' annunciator. However, neither [green diamond][ON] nor [2nd][ON] will turn the calculator off. Pressing [ON] to break the program doesn't work, either. The calculator must be reset with [2nd][hand] + [ON].

This bug *does not* occur if *return* is used instead of *stop*.

The bug also occurs if *expr()* is used in a program, to execute the archived program containing the *stop* instruction. For example, this program

```
stoptry()
prgm
expr("stopit()")
endprgm
```

will also cause the bug, but only if *stopit()* is archived.

The bug will *not* occur if the program containing the stop instruction is not archived.

Note that the bug also occurs if *stopit()* is called indirectly using *expr(),* like this:

```
app1()
prgm
expr("stoptry()")
endprgm
```

```
stoptry()
prgm
stopit()
endprgm

stopit()
prgm
stop
endprgm
```

In this case, *app1()* is not archived, but *stoptry()* and *stopit()* are archived, and the bug occurs. And, in this case, *stopit()* is not called with *expr(),* but the routine that calls *stopit()* does use *expr().*

This bug is annoying because it can prevent you from implementing desired program operation. I have a complex application that uses an 'exit' program to clean up before the user quits the program. Clean-up includes deleting global variables, resetting the user's mode settings, and restoring the current folder at the time the application was called. This 'exit' routine is called from the application mainline, so I would like to use the *stop* instruction to terminate operation. There are several obvious solutions:

1. Leave the 'exit' routine unarchived. This consumes RAM for which I have better uses.
2. Call the 'exit' routine directly, without *expr().* This prevents me from using an application launcher I wrote to manage my apps.
3. Archive most of the exit program, but put the final *stop* instruction in its own, unarchived program which is called by the exit program. While this seems like an acceptable work-around, it should not be necessary.
4. Call the 'exit' routine as usual, but use *return* instead of *stop* to terminate program operation.