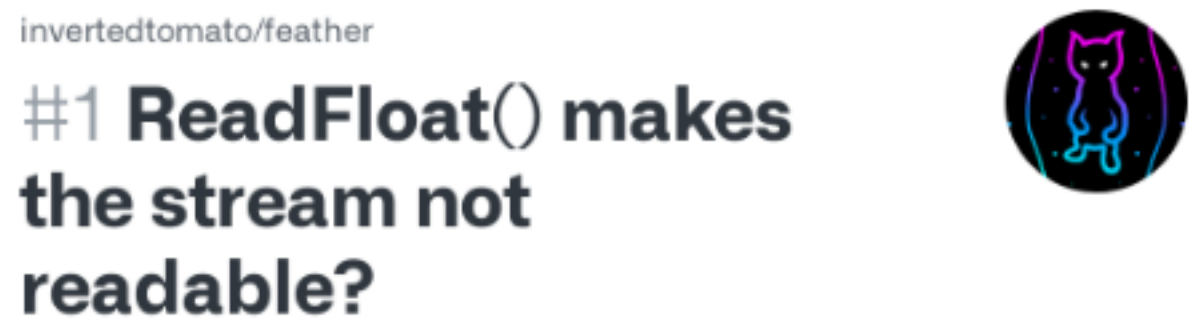
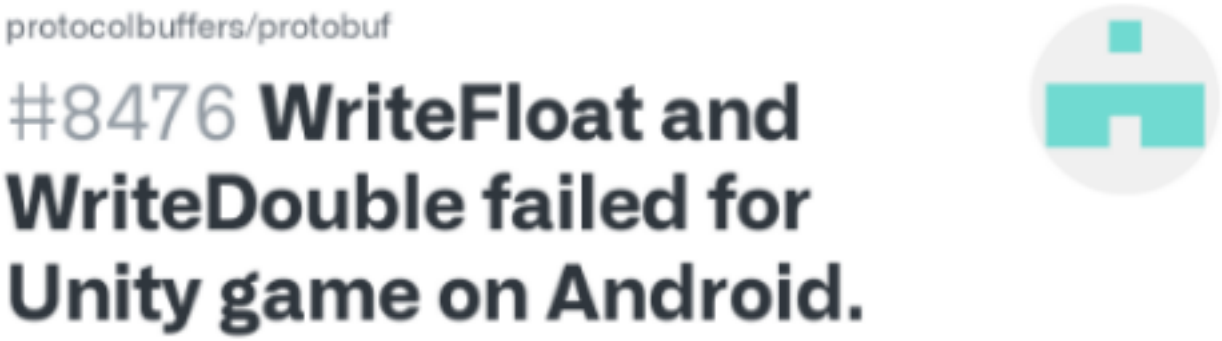
Floating-point input-output procedures

The passage you sent describes two procedures for floating-point input/output in assembly language:

- **ReadFloat:** Reads a floating-point value from the keyboard and pushes it on the floating-point stack.



- **WriteFloat:** Writes the floating-point value at ST(0) to the console window in exponential format.



The ReadFloat procedure accepts a wide variety of floating-point formats, including:

```
35
+35.
-3.5
.35
3.5E5
3.5E005
-3.5E+5
3.5E-4
+3.5E-4
```

The WriteFloat procedure writes the floating-point value at ST(0) to the console window in exponential format.

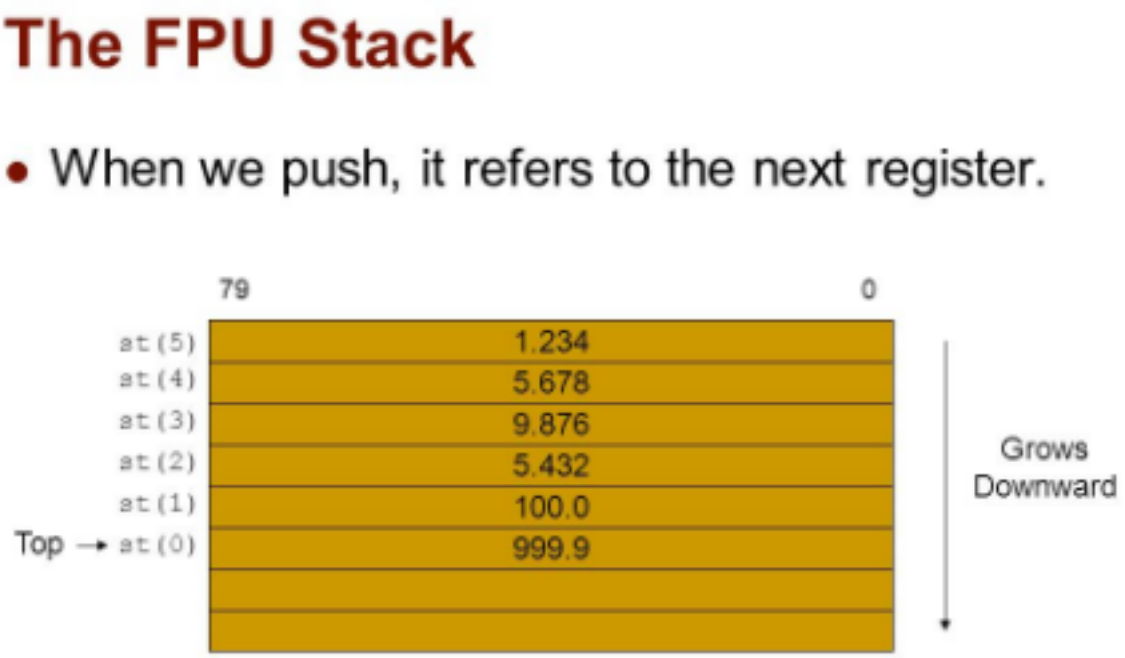
Here's the example program that demonstrates the use of the ReadFloat and WriteFloat procedures in assembly language. This program pushes two floating-point values onto the FPU stack, displays the FPU stack, takes user input for two values, multiplies them, and displays their product. I'll provide you with the assembly code:

```
; 32-bit Floating-Point I/O Test (floatTest32.asm)
INCLUDE Irvine32.inc
INCLUDE macros.inc
.data
    first REAL8 123.456
    second REAL8 10.0
    third REAL8 ?

.code
    main PROC
        finit                   ; Initialize FPU
        ; Push two floats and display the FPU stack.
        fld first               ; Push the first value onto the FPU stack.
        fld second              ; Push the second value onto the FPU stack.
        call ShowFPUStack       ; Display the FPU stack.
        ; Input two floats and display their product.

        mWrite "Please enter a real number: "
        call ReadFloat          ; Read the first floating-point number.
        mWrite "Please enter a real number: "
        call ReadFloat          ; Read the second floating-point number.

        fmul                    ; Multiply ST(0) by ST(1).

        mWrite "Their product is: "
        call WriteFloat         ; Display the product.
        call Crlf               ; Add a line break.

        exit
    main ENDP
END main
```

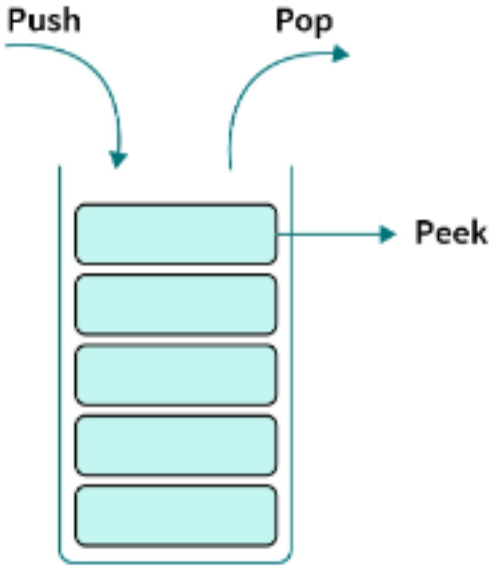
This assembly code is designed to demonstrate the use of floating-point input and output procedures while performing basic arithmetic operations on these floating-point values using the FPU (Floating-Point Unit). Let's break down the code step by step.

Initialization (finit): The program begins by initializing the FPU using the finit instruction. This is a necessary step to prepare the FPU for floating-point operations.

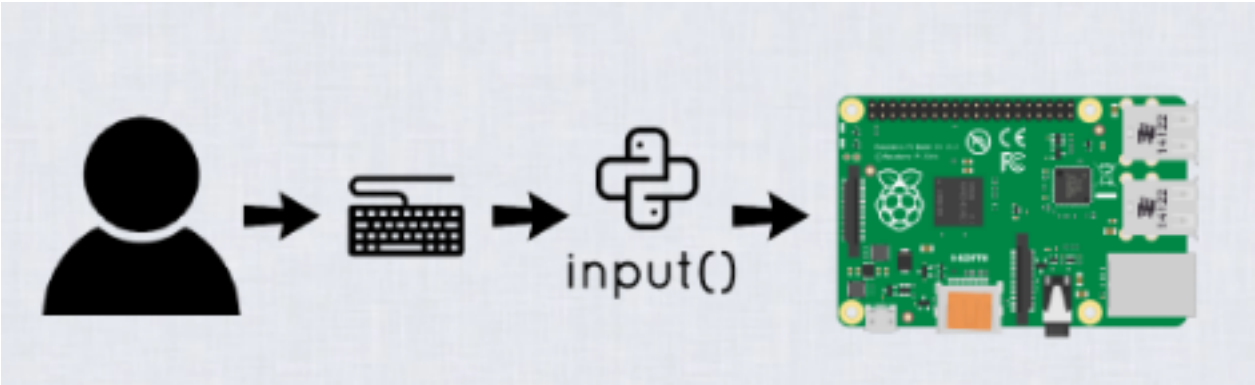
Pushing Values onto the FPU Stack: Two floating-point values are pushed onto the FPU stack. These values are stored in memory as first and second. The fld (floating-point load) instructions are used to load these values onto the FPU stack. fld first pushes the value of first onto the stack, and fld second pushes the value of second onto the stack.



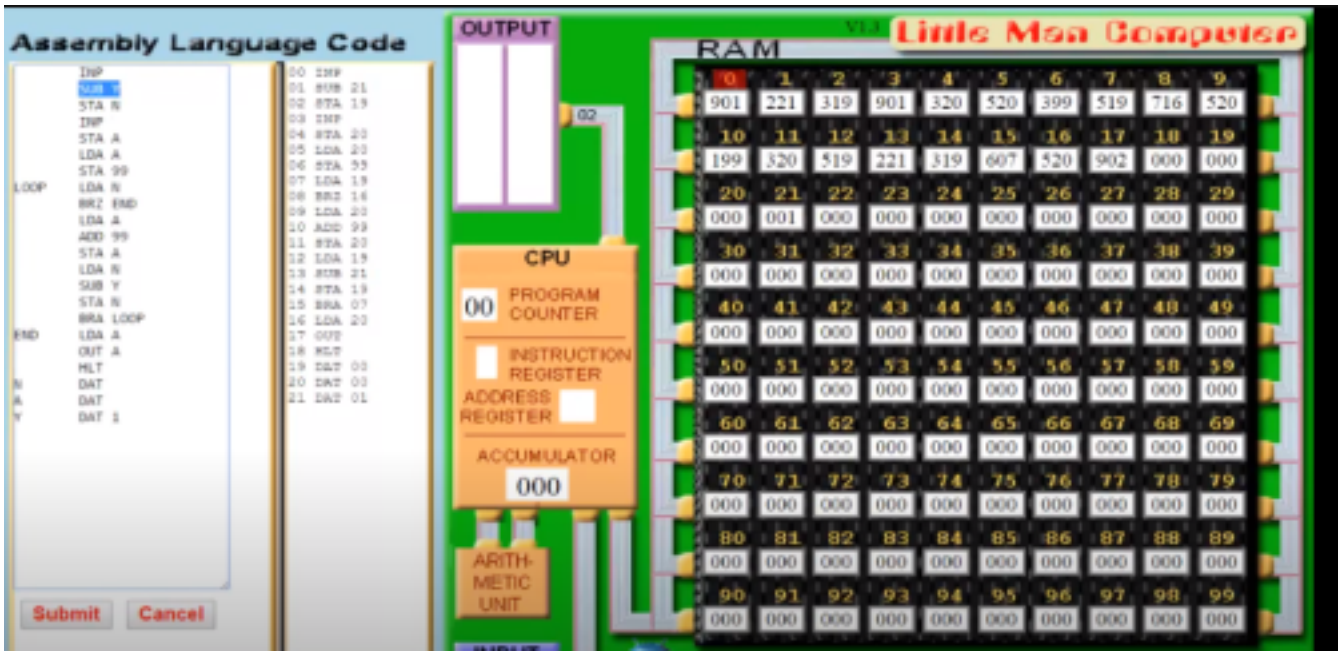
Displaying the FPU Stack: After pushing the values onto the FPU stack, the program calls a custom procedure called ShowFPUStack. This procedure is responsible for displaying the contents of the FPU stack. It helps visualize the values stored on the stack at this point.



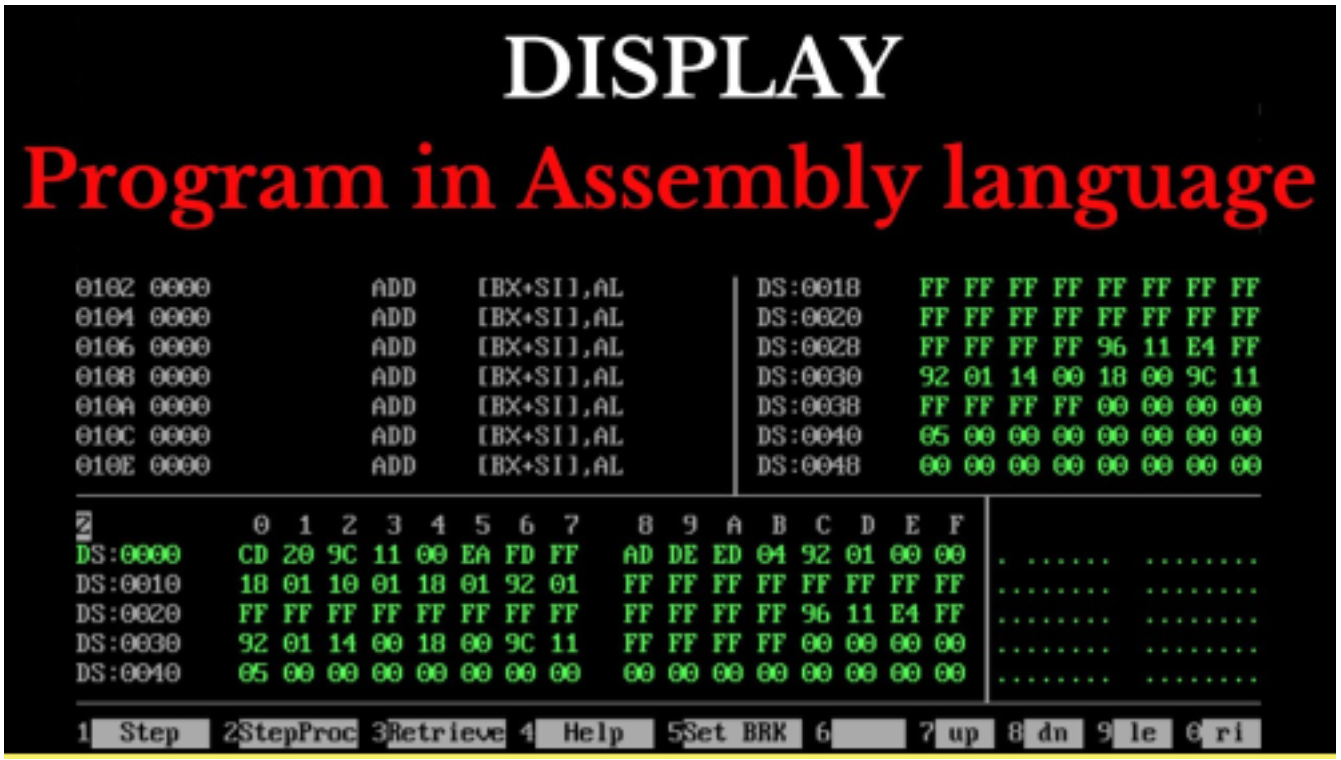
User Input: The program prompts the user to enter two real numbers. It uses the mWrite function to display the input prompt. Then, it calls the ReadFloat procedure, which reads the user's input as a floating-point number. This process is repeated for the second number.



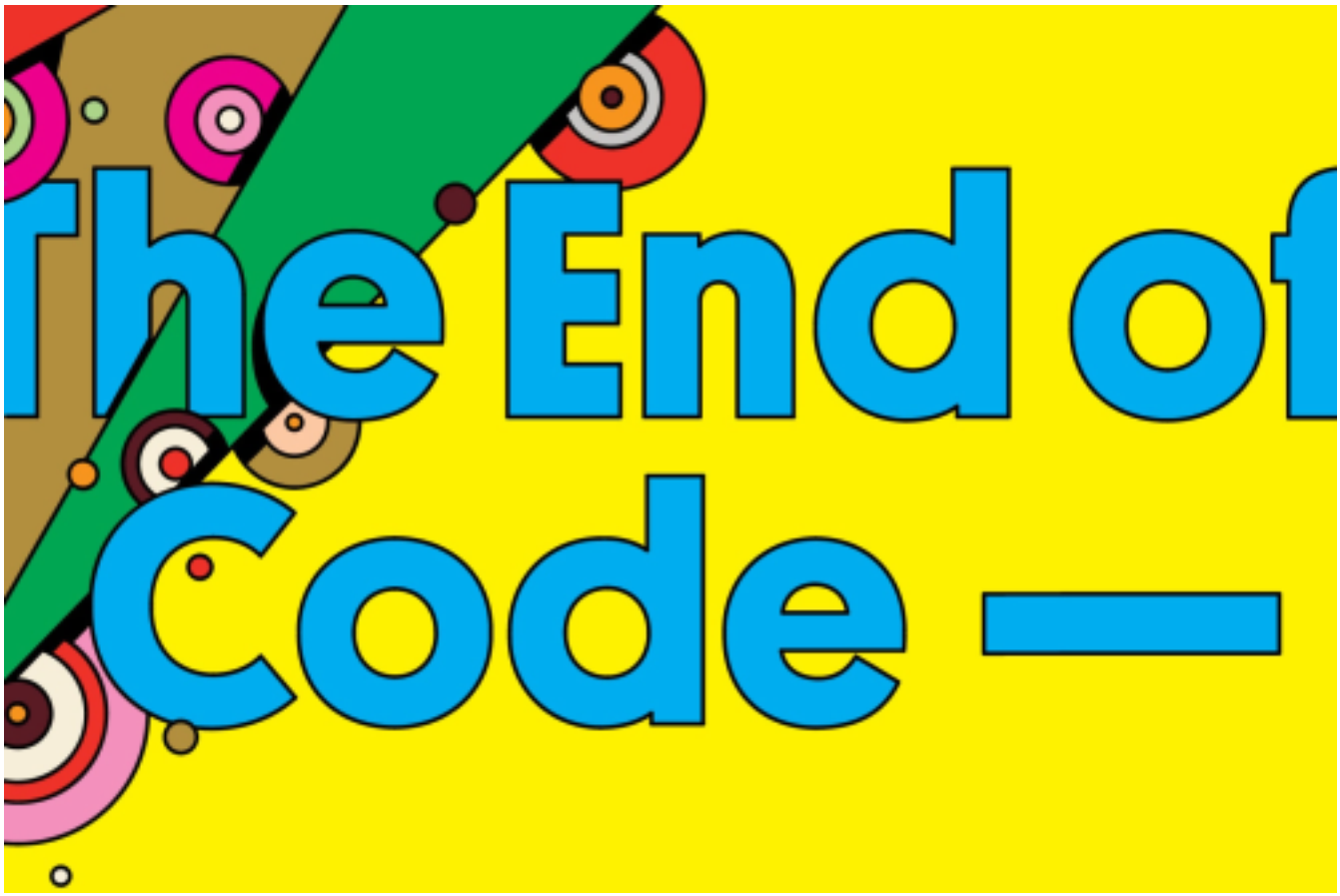
Multiplication (fmul): Once both user inputs are on the FPU stack, the program uses the fmul instruction to multiply these values. The fmul instruction multiplies the value on top of the stack (ST(0)) by the next value (ST(1)) and stores the result in ST(0). In this case, it effectively calculates the product of the two numbers entered by the user.



Displaying the Result: After the multiplication is performed, the program uses mWrite to display the text "Their product is: " to the console. Then, it calls the WriteFloat procedure to display the result of the multiplication in exponential format.



End of Program: Finally, the program adds a line break using Crlf for a clean console output and exits.



The primary purpose of this program is to showcase the handling of floating-point values, input, and output, as well as basic arithmetic operations on these values using the FPU.

The FPU stack is crucial in managing these floating-point values during the operations, and the code illustrates the sequence of actions involved in working with the FPU for floating-point calculations.