

**Pseudo**  
**FPU**<sup>1.1</sup>  
**Programmer Information**

John M. Neil  
3257 Greer Road  
Palo Alto, California 94303

January 7, 1991

## **What PseudoFPU Does**

PseudoFPU is a cdev/INIT which allows programs that expect a Motorola 68881 or 68882 Floating Point Unit (FPU) to work correctly without one. It is primarily intended for use with the Macintosh IIsi and Macintosh LC, which do not contain an FPU. PseudoFPU works by translating FPU instructions to SANE calls. Since both SANE and the FPU conform to the IEEE Standard for floating point arithmetic, the differences between the hardware FPU and PseudoFPU are minimal. The current known incompatibilities with a hardware FPU are:

- FRESTORE does not support the busy state frame.
- FMOD produces the same result as FREM.
- Mid-instruction exceptions are reported as post-instruction exceptions.
- If an exception occurs in trace mode, two instructions will execute before control returns to the debugger.
- Code which puts data below the stack pointer and then issues an FPU instruction will not work. This of course is

an extreme no-no since data below the stack pointer can be clobbered by interrupt routines as well.

- Some emulated FPU instructions produce more accurate results than on a hardware FPU. See page 197 of the *Apple Numerics Manual, 2nd Edition* for a list of these instructions. These differences can be exploited by software to detect PseudoFPU. The functions listed at the bottom of page 198 are emulated using algorithms in Chapter 9 of the *Apple Numerics Manual*.

The current list of known application incompatibilities are:

- Any program which patches the F-line exception vector itself will not work with PseudoFPU. A development environment application would need to check for the presence of PseudoFPU and patch a location inside the emulator to trap F-line exceptions. That location is 2 + the value of 68000 F-line exception vector.
- Programs using recursive algorithms may have problems. PseudoFPU can require up to 600 bytes of stack space per FPU instruction. Recursive programs will probably require more stack space than usual to work properly, if they will work at all.

- A bug in the MPW 3.1 `nan()` function means that any program calling `nan()` will not work. Because of this, the MPW functions `atan2()`, `asin()`, and `acos()` will not work when they try to produce QNaNs. Also, `fscanf` will not read in NaNs correctly. The bug has been fixed in MPW 3.2.
- Code which assumes that calls to SANE affect the FPU will not work. The MPW functions `sinh()` and `cosh()` may produce incorrect results in the exception status register, as the library code makes this assumption.

PseudoFPU requires about 16K of system heap space to install correctly.

## **Performance**

As you might imagine, all is not gravy with PseudoFPU. Performance ranges from 30% to 85% of what one gets calling SANE directly, depending on the instruction type. However, applications typically do not slow down this much since they do not execute FPU instructions 100% of the time.

With PseudoFPU installed, applications using direct FPU calls obtain a different performance tradeoff than using SANE. Take the performance of SANE on a non-FPU machine as a benchmark. SANE works about 10 times faster on machines with FPUs. Direct FPU calls run about 100 times faster on machines with FPUs, and 30-85% of SANE on non-FPU machines.

## **SysEnviroms/Gestalt Patch**

The emulator patches Gestalt to indicate the presence of an

FPU to applications. While this maximizes the number of programs that work, it would slow down the few programs that can perform either SANE or FPU calls because they will use the software FPU instead of SANE. The only known applications of this type are DataDesk and Excel. The Gestalt patch checks to see if the current application is one of these two, and if so reports no FPU present. These two applications will therefore run at maximum speed regardless of whether PseudoFPU is installed or not.