New Technical Notes
Macintosh

Developer Support

OS 4 - OmegaSANE
Operating System

Written by: Dave Radcliffe and Colin McMaster May 1992

System 7.0.1 introduced a new version of SANE (the Standard Apple Numerics Environment) known as OmegaSANE. This Note discusses the features of OmegaSANE and the associated compatibility risks. This note covers:

Topics
- OmegaSANE features, including:
  - Correctly rounded binary ↔ decimal conversions
  - Faster transcendental functions
  - Backpatching of Pack 4 SANE traps for faster package entry
- Compatibility risks due to backpatching

Introduction

System 7.0.1 introduced a new version of the Standard Apple Numerics Environment (SANE) package referred to as OmegaSANE (ΩSANE). While it improves the performance of SANE, it is not without compatibility risks, which are detailed below. New binary ↔ decimal conversions have been included that are correctly rounded across the entire range of double extended. This will result in incompatibility with previous conversion algorithms for a variety of input values; the results of the ΩSANE conversions are uniformly more accurate, and will be compatible with future releases of SANE.

ΩSANE Features

The ΩSANE release provides a number of performance enhancements while maintaining conformance with SANE. The following enhancements have been implemented:

- Correctly rounded binary ↔ decimal conversions
- Faster transcendental functions
- Backpatching of SANE traps for faster package entry

Tables showing the performance of ΩSANE are given below. A key aspect of the performance gain is the “backpatching” of SANE traps. This mechanism will be discussed below under “Pitfalls.”

The version of ΩSANE supplied with System 7.0.1 installs only on machines equipped with a Macintosh IIci ROM or later and also equipped with an FPU. For example, it installs on a Macintosh IIsi running System 7.0.1 and equipped with an FPU, but not on one without an FPU. On the Macintosh Quadras and the PowerBook 140/170 it is in ROM, although it doesn't load on the PowerBook 140 unless it has an optional FPU. On machines where the FPU is optional, addition of an FPU may require re-installation of System 7.0.1 before the FPU version of ΩSANE will load.

Table 1 presents configuration information concerning the versions of SANE which may be installed by System 7.0.1. Information in this table is subject to change. The FPU column states whether the machine has an FPU or if it is optional. The column labeled "Correctly Rounded Bin ↔ Dec" shows whether improved versions of the binary ↔ decimal conversion routines (discussed below) are available, and which version (V.1 or V.2) is supplied. The "ΩSANE FPU Version" column states whether ΩSANE will load on a given configuration.

Note: The information in this table will undoubtedly change in the future. Developers should not assume that any particular machine/system software combination does or does not have ΩSANE.

Table 1 SANE Configurations

Macintosh CPU	FPU	Correctly Rounded Bin↔Dec	ΩSANE FPU Version
Mac Plus	No	No	No
Mac SE	No	No	No
Mac Classic	No	No	No
Mac Classic II	Optional	V.2 in ROM	w/ 7.0.1 & FPU†
Mac LC	Optional	V.1 in ROM	w/ 7.0.1 & FPU†
Mac IIsi	Optional	V.1 in ROM	w/ 7.0.1 & FPU†
Mac SE/30	Yes	No	No
Mac II	Yes	No	No
Mac IIx	Yes	No	No
Mac IIcx	Yes	No	No
Mac IIci	Yes	V.2 with 7.0.1	w/ 7.0.1
Mac IIfx	Yes	V.2 with 7.0.1	w/ 7.0.1
PowerBook 100	Yes	No	No
PowerBook 140	Optional	V.2 in ROM	w/ 7.0.1 & FPU†
PowerBook 170	Yes	V.2 in ROM	Yes
Quadra 700	Yes	V.2 in ROM	Yes
Quadra 900	Yes	V.2 in ROM	Yes

† FPU optional systems may require reinstallation of System 7.0.1 after installation of an FPU before the ΩSANE FPU version will load.

There is no way programmatically to disable ΩSANE in System 7.0.1, nor is it a user option. There is also no way to reliably detect that ΩSANE is present.

Binary ↔ Decimal Conversions

ΩSANE includes binary ↔ decimal conversion routines that may produce results that differ from previous versions of SANE. Versions of these routines have been in some machine ROMs since the Macintosh IIsi, and an improved version (V.2) is in the newest ROMs, as well as ΩSANE. Refer to Table 1 for current configuration information. As a result of these improved conversion routines, floating-point constants that are used in the body of source code will compile differently in the presence of ΩSANE than with older SANE engines. The results are uniformly better, but may cause unexpected variances from test suites (for instance). Therefore, care must be given to the arithmetic environment in which compilations are made. One can tell which version of the binary↔decimal conversions is currently in use by performing the following computation on the Calculator DA: 45/100 – 0.45. On older versions of SANE, the result is –2.71051E-20. With ΩSANE, the result is 0 as expected.

Performance Improvements

ΩSANE can significantly improve the performance of many floating-point SANE operations. It does this by replacing _FP68K trap calls with JSR instructions directly into the appropriate SANE code. This is discussed in more detail under “Pitfalls.” 7.0.1 ΩSANE does not alter _Elems68K trap calls, but internal code improvements are used to increase the performance of transcendental functions. Finally, ΩSANE does not affect the performance of code compiled to use the FPU directly, although some library routines used with such code (such as the CSANELib881.o routines NextAfter, Classify, Scalb, and Remainder) will be backpatched by ΩSANE because they call _FP68K.

Table 2 shows typical performance improvement using ΩSANE on a Macintosh IIci. Of course, actual performance depends on how heavily you use SANE and the mixture of SANE operations you use.

Table 2 SANE Performance

Operation	Macintosh IIci SANE (in flops)	Macintosh IIci ΩSANE (in flops)	Speedup Factor
Add/Subtract	18,794	59,259	3.15
Multiply	18,182	53,571	2.95
Divide	18,476	55,300	2.99
Square Root (Exact)	18,547	65,934	3.55
Square Root (Inexact)	18,349	59,113	3.22
Cosine	902	7,058	7.82
Sine	1,290	8,108	6.29
Tangent	826	7,185	8.70
Logarithm	820	7,643	9.32
Exponential	723	7,317	10.12
Compound	413	3,324	8.05
Annuity	358	3,191	8.91

Pitfalls

ΩSANE achieves most of its performance improvement through use of a technique known as "backpatching." Use of backpatching introduces a number of compatibility implications. To implement backpatching, the front end of toolbox SANE has been altered to have multiple entry points corresponding to the most important operations, and *your* RAM-based application code is modified *on the fly* replacing traps with speedier JSRs to the appropriate entry point. Subsequent execution causes the code to bypass the trap dispatcher and call SANE directly. For example, an archetypal SANE package call looks like this:

```
        pea    fooSrc ; Push address of the source operand
        pea    barDst ; Push the address of the destination operand
        move.w #OpCode,-(sp) ; Push a SANE opcode word
        _FP68K       ; Trap to SANE
```

Notice that the last two instructions occupy three words, just enough for the desired JSR. ΩSANE modifies the RAM-image of the application resulting in code like the following:

```
        pea    fooSrc              ; Push address of the source operand
        pea    barDst              ; Push the address of the dest. operand
        jsr    'SANE entry point'  ; JSR to appropriate SANE entry point
```

Herein lies a potential problem. There is nothing to guarantee that the instruction immediately prior to the _FP68K trap was actually executed. A program control operation, such as a BRA, could have caused control to be passed to the _FP68K operation without executing the preceding move.w. For instance, an execution sequence such as the following:

```
        pea    subSrc        ; Push source for subtract
        pea    subDst        ; Push destination for subtract
        move   #$0002,-(sp)  ; Push subtract extended opcode word
        bra    @1            ; Branch to _FP68K trap
        .
        .
        .
        pea    addSrc2       ; Push source for add
        pea    addDst2       ; Push destination for add
        move   #$0000,-(sp)  ; Push add extended opcode word
@1      _FP68K               ; Trap to SANE
```

would have the unseemly result of "backpatching" a JSR to the extended add entry point, and future executions of the subtraction code would branch to the middle of a JSR causing an illegal instruction error (the illegal instruction error might not occur right away). Note, however, that the following similar code sequence works correctly in the presence of backpatching:

```
        pea    addSrc        ; Push source for add
        pea    addDst        ; Push destination for add
        bra    @1            ; Branch to _FP68K trap
        .
        .
        .
        pea    addSrc2       ; Push source for add
        pea    addDst2       ; Push destination for add
@1      move   #$0000,-(sp)  ; Push add extended opcode word
        _FP68K               ; Trap to SANE
```

The type of code sequence that is problematic for backpatching (as above) cannot be emitted by the current MPW C and Pascal compilers or by current Think™ compilers. It also cannot occur with code generated using the macros contained in SANEMacs.a.

Warning: Although the current generation of MPW compilers create ΩSANE-safe code, the earlier, MPW C 2.0.2 compiler may, under limited circumstances, generate code that has problems with ΩSANE. This is discussed in more detail below.

Additionally, if you have hand coded assembly or code which has been otherwise optimized to use common _FP68K trap instructions you may be at risk from backpatching. We recommend you adjust your code to conform to the prototypical sequence above as there is no performance penalty involved.

Another common programming practice is illustrated in the following example:

```
        move.w #$0000,D0     ; Move extended add opcode word to D0
        bra    @1            ; Branch to shared backend for SANE trap call
        .
        .
        .
        move.w #$0002,D0     ; Move extended subtract opcode word to D0
        bra    @1            ; Branch to shared backend for SANE trap call
        .
        .
        .
@1      move.w D0,-(SP)      ; Push SANE opcode word on stack
        _FP68K               ; Trap to SANE
```

This code operates correctly in the presence of ΩSANE, but is clearly not backpatchable. If you have code that does this, you might consider rewriting it to obtain the performance increase, but the code will work as is.

MPW C 2.0.2

The MPW C 2.0.2 compiler may, under limited circumstances, generate code that works incorrectly under ΩSANE. Apple recommends that all developers use the latest development tools, but as conversion of source may be difficult and time consuming in some cases, below is a description and workaround for the problem. Again, this only affects code compiled without the -mc68881 option.

The following code demonstrates the problem

```
struct foo {
        double data;
        float num;
};

foobar(f,b)
struct foo *f;
unsigned char b;
{
        extended num;
```

```
        num = 3.14159;
        if (b)
                f->data = num;      /* FP operation ends block */
        else
                f->num = num;       /* Another FP operation ends block */

        return;
}
```

The critical combination of events occurs when more than one block in an if-else or switch statements ends with a floating-point operation or conversion. The compiler tries to optimize the final floating-point operation by pushing a selector on the stack then branching to a common `_FP68K` trap. This is a classic example of code, like that cited above, that works incorrectly under ΩSANE.

The workaround is to eliminate the final floating-point operation. Here is one way to do this:

```
foobar(f,b)
struct *f;
unsigned char b;
{
        extended num;
        int idummy;
        num = 3.14159;
        if (b) {
                f->data = num;
                idummy = 1;   /* End if block with a non-FP operation */
        } else {
                f->num = num;
                idummy = 0;   /* End else block with a non-FP operation */
        }

        return;
}
```

In this case, each floating-point operation gets its own `_FP68K` trap so there are no problems.

Other Pitfalls

The backpatching performed by ΩSANE can have other implications for developers. For example, applications that checksum their code segments (perhaps to check for viruses) will detect that the segment has been modified. Such applications should checksum segments only at application startup or when the segments are loaded, not after they have been executed.

Likewise, an application must not create a situation which will cause the Resource Manager to write a code segment out to disk after it has been executed. If such a segment been modified by ΩSANE, it can fail when subsequently loaded into memory. Since `CloseResFile` is called when the application quits it will automatically write out changed resources (i.e. if the `resChanged` attribute is true) when closing the file.

Further Reference:

- Apple Numerics Manual, Second Edition