

Technote 1158

Levatatus Interruptus*

By George Warner
Apple Worldwide Developer Technical Support

(*This has nothing to do with why cartoon characters who run off cliffs don't fall until they look down.)

CONTENTS

[The Problem: Volatile FP regs not saved](#)

[An Example](#)

[Disassembly and Analysis](#)

[The Glue Code](#)

[New \(and Improved!\) Example](#)

[New \(and Improved!\) Analysis](#)

[New \(and Improved!\) Glue Code](#)

[Summary](#)

[Downloadables](#)

The use of floating point instructions is usually handled by a development system's libraries. One special case not properly handled by these libraries is the saving and restoring of the volatile floating point registers for interrupt routines. Floating-point assembly code will need to be written for this case. This Technote will address this topic.

This Technote is of interest to any developers writing PowerPC interrupt routines that use floating point registers.

The Problem: Volatile FP registers not saved.

The PowerPC interrupt dispatcher saves and restores the volatile general purpose registers (GPR0-12) but not the volatile floating point registers (FPR0-13)*. Because of this, any PowerPC interrupt routine that uses any of the volatile floating point registers is responsible for saving and restoring them; Otherwise, any routines that get interrupted will unexpectedly have these registers changed by the interrupting code.

* All non-volatile registers are saved and restored in the prolog and epilog of the interrupt routine. This code is automatically generated by the compiler.

An Example:

Here is an example of an interrupt routine that uses (volatile and non-volatile) floating point registers:

```
pascal void My_SInterruptProc(SPBPtr inParamPtr, Ptr dataBuffer, SInt16 peakAmplitude,
SInt32 sampleSize)
{
    float tPIE1 = 0.0f;
    float tPIE2 = 0.0f;
    Ptr tPtr = dataBuffer;
    SInt32 count = inParamPtr->bufferLength;

    while (count-->0)
    {
        *tPtr++ ^= 0x80;
        tPIE1 += (tPIE1 + 1.0f) * 1.1f;
        tPIE2 += (tPIE1 + 1.0f) * 1.1f;
    }
}
```

Analysis:

Here is the disassembly and analysis of this interrupt routine:

```

pascal void My_SInterruptProc(SPBPtr inParamPtr, Ptr dataBuffer, SInt16 peakAmplitude,
SInt32 sampleSize)
{
00000000: DBE1FFF8 stfd    fp31, -8(SP)           // save volatile float regs (fp30-31)
00000004: DBC1FFF0 stfd    fp30, -16(SP)
00000008: 93E1FFEC stw     r31, -20(SP)           // save volatile GP regs (r30-31)
0000000C: 93C1FFE8 stw     r30, -24(SP)
        float tPIE1 = 0.0f;
00000010: C3E20000 lfs     fp31, @1360(RTOC) // load volatile float regs
        float tPIE2 = 0.0f;
00000014: C3C20000 lfs     fp30, @1360(RTOC)
        Ptr tPtr = dataBuffer;
00000018: 7C9F2378 mr      r31, r4             // load pointer to buffer
        SInt32 count = inParamPtr->bufferLength;
0000001C: 83C3000C lwz    r30, 12(r3)
        while (count--)
        {
00000020: 48000038 b        *+56                ; $00000058
                *tPtr++ ^= 0x80;
00000024: 7FE5FB78 mr      r5, r31
00000028: 3BFF0001 addi   r31, r31, 1
0000002C: 88050000 lbz    r0, 0(r5)
00000030: 6800FF80 xori   r0, r0, $ff80
00000034: 98050000 stb    r0, 0(r5)
                tPIE1 += (tPIE1 + 1.0f) * 1.1f;
00000038: C0220000 lfs     fp1, @1361(RTOC) // NOTICE: using non-volatile float (fp1)
0000003C: C0020000 lfs     fp0, @1362(RTOC) // " (fp0)
00000040: EC00F82A fadds  fp0, fp0, fp31 // " (fp0)
00000044: EFE1F83A fmadds  fp31, fp1, fp0, fp31 // " (fp0-1)
                tPIE2 += (tPIE1 + 1.0f) * 1.1f;
00000048: C0220000 lfs     fp1, @1361(RTOC) // " (fp1)
0000004C: C0020000 lfs     fp0, @1362(RTOC) // " (fp0)
00000050: EC00F82A fadds  fp0, fp0, fp31 // " (fp0)
00000054: EFC1F03A fmadds  fp30, fp1, fp0, fp30 // " (fp0-1)
        }
00000058: 2C1E0000 cmpwi  r30, 0             // test count
0000005C: 3BDEFFFF subi  r30, r30, 1       // bump count
00000060: 4082FFC4 bne    *-60 ; $00000024 // (loop till zero)

00000064: CBE1FFF8 lfd    fp31, -8(SP)           // restore volatile float regs (fp30-31)
00000068: CBC1FFF0 lfd    fp30, -16(SP)
0000006C: 83E1FFEC lwz    r31, -20(SP)           // restore volatile GP regs (r30-31)
00000070: 83C1FFE8 lwz    r30, -24(SP)
00000074: 4E800020 blr                    // return (branch to link register)
}

```

The problem here is that the non-volatile floating point registers (fp0-1) were not saved/restored by the prolog/epilog code. Any code interrupted by this routine would have unexpected values suddenly show up in fp0-1.

The Glue Code:

This technote proposes a solution which is to write PowerPC assembly code that saves and restores the volatile floating point registers. The totally paranoid approach is to save/restore ALL the volatile FP regs. (Note that the non-volatile FP regs (14-31) are saved in your routines prolog/epilog code.) If your routine only uses the volatile FP regs 0 and 1 (like the example did) you should reduce the struct, save and restore to ONLY define, and save and restore those registers. DON'T delete the line to define, save and restore the FP SCR. You always want to save and restore it. Here is the paranoid solution:

```
#if TARGET_CPU_PPC
typedef struct FPUREgs
{
    float    FPR00;
    float    FPR01;
    float    FPR02;
    float    FPR03;
    float    FPR04;
    float    FPR05;
    float    FPR06;
    float    FPR07;
    float    FPR08;
    float    FPR09;
    float    FPR10;
    float    FPR11;
    float    FPR12;
    float    FPR13;
    float    FPSCR;
} FPUREgsRec, *FPUREgsPtr, **FPUREgsHdl;

// function prototypes
asm void SavePowerPCVolatileFPRs(FPUREgsPtr pFPUREgsPtr);
asm void RestorePowerPCVolatileFPRs(FPUREgsPtr pFPUREgsPtr);

//
// Save the non-volatile PowerPC floating pointer registers
// into the passed in structure.
//

asm void SavePowerPCVolatileFPRs(FPUREgsPtr pFPUREgsPtr)
{
    stfd    fp0, FPUREgs. FPR00(r3)
    stfd    fp1, FPUREgs. FPR01(r3)
    stfd    fp2, FPUREgs. FPR02(r3)
    stfd    fp3, FPUREgs. FPR03(r3)
    stfd    fp4, FPUREgs. FPR04(r3)
    stfd    fp5, FPUREgs. FPR05(r3)
    stfd    fp6, FPUREgs. FPR06(r3)
    stfd    fp7, FPUREgs. FPR07(r3)
    stfd    fp8, FPUREgs. FPR08(r3)
    stfd    fp9, FPUREgs. FPR09(r3)
    stfd    fp10, FPUREgs. FPR10(r3)
    stfd    fp11, FPUREgs. FPR11(r3)
    stfd    fp12, FPUREgs. FPR12(r3)
    stfd    fp13, FPUREgs. FPR13(r3)

    mffs    fp0                                // copy fpscr into fp0
    stfd    fp0, FPUREgs. FPSCR(r3)           // save fpscr
    blr
}

```

```
//  
// Restore the non-volatile PowerPC floating pointer registers  
// from the passed in structure.  
//  
asm void RestorePowerPCVolatileFPRs(FPURegsPtr pFPURegsPtr)  
{  
    lfd    fp0,  FPURegs.FPSCR(r3)           // load saved fpSCR into fp0  
    mtfsf  0xff, fp0                        // move it back into the fpSCR  
  
    lfd    fp0,  FPURegs.FPR00(r3)  
    lfd    fp1,  FPURegs.FPR01(r3)  
    lfd    fp2,  FPURegs.FPR02(r3)  
    lfd    fp3,  FPURegs.FPR03(r3)  
    lfd    fp4,  FPURegs.FPR04(r3)  
    lfd    fp5,  FPURegs.FPR05(r3)  
    lfd    fp6,  FPURegs.FPR06(r3)  
    lfd    fp7,  FPURegs.FPR07(r3)  
    lfd    fp8,  FPURegs.FPR08(r3)  
    lfd    fp9,  FPURegs.FPR09(r3)  
    lfd    fp10, FPURegs.FPR10(r3)  
    lfd    fp11, FPURegs.FPR11(r3)  
    lfd    fp12, FPURegs.FPR12(r3)  
    lfd    fp13, FPURegs.FPR13(r3)  
   blr  
}  
#endif // #if TARGET_CPU_PPC
```

New (and Improved!) Example:

If we add lines to the example to call our routines to save and restore the volatile floats it should now look like this:

```

pascal void My_SIIInterruptProc(SPBPtr inParamPtr, Ptr dataBuffer, SInt16 peakAmplitude,
SInt32 sampleSize)
{
  #if TARGET_CPU_PPC
    FPUREgsRec tFPUREgsRec;
    SavePowerPCVolatil eFPRs(&tFPUREgsRec);
  #endif

    float tPIE1 = 3.14159f;
    float tPIE2 = 3.14159f;

    Ptr tPtr = dataBuffer;
    SInt32 count = inParamPtr->bufferLength;
    while (count-->0)
    {
      *tPtr++ ^= 0x80;
      tPIE1 += (tPIE1 + 1.0f) * 1.1f;
      tPIE2 += (tPIE1 + 1.0f) * 1.1f;
    }
  #if TARGET_CPU_PPC
    RestorePowerPCVolatil eFPRs(&tFPUREgsRec);
  #endif
}

```

New (and Improved!) Analysis:

Here is the disassembly and analysis of the new (and improved) routine:

```

pascal void My_SIIInterruptProc(SPBPtr inParamPtr, Ptr dataBuffer, SInt16 peakAmplitude,
SInt32 sampleSize)
{
    FPUREgsRec tFPUREgsRec;
00000000: 7C0802A6 mflr    r0           // move link register to R0
00000004: DBE1FFF8 stfd    fp31, -8(SP) // save volatile float regs (fp30-31)
00000008: DBC1FFF0 stfd    fp30, -16(SP)
0000000C: 93E1FFEC stw     r31, -20(SP) // save volatile GP regs (r30-31)
00000010: 93C1FFE8 stw     r30, -24(SP)
00000014: 90010008 stw     r0, 8(SP)    // save link register
00000018: 9421FF70 stwu   SP, -144(SP) // save stack pointer
0000001C: 906100A8 stw     r3, 168(SP) // save first two passed parameters
00000020: 908100AC stw     r4, 172(SP)
    SavePowerPCVolatil eFPRs(&tFPUREgsRec);
    {
00000024: 38610038 addi   r3, SP, 56 // point r3 at tFPUREgsRec on stack
00000028: 48000001 bl     .SavePowerPCVolatil eFPRs // call our save routine

        float tPIE1 = 0.0f;
0000002C: C3E20000 lfs    fp31, @1360(RTOC) // load volatile float regs
        float tPIE2 = 0.0f;
00000030: C3C20000 lfs    fp30, @1360(RTOC)
        Ptr tPtr = dataBuffer; // load pointer to buffer
00000034: 83E100AC lwz    r31, 172(SP)
        SInt32 count = inParamPtr->bufferLength;
00000038: 806100A8 lwz    r3, 168(SP)
0000003C: 82C20000 lwz    r30, 172(SP)

```

```

0000003C: 85C5000C lwz      r30, 12(r3)
                while (count--)
                {
00000040: 48000038 b          *+56                ; $00000078
                *tPtr++ ^= 0x80;
00000044: 7FE3FB78 mr        r3, r31
00000048: 3BFF0001 addi     r31, r31, 1
0000004C: 88030000 lbz     r0, 0(r3)
00000050: 6800FF80 xori    r0, r0, $ff80
00000054: 98030000 stb     r0, 0(r3)
                tPIE1 += (tPIE1 + 1.0f) * 1.1f;
00000058: C0220000 lfs     fp1, @1361(RTOC) // NOTICE: using non-volatile float (fp1)
0000005C: C0020000 lfs     fp0, @1362(RTOC) // " (fp0)
00000060: EC00F82A fadds   fp0, fp0, fp31 // " (fp0)
00000064: EFE1F83A fmadds  fp31, fp1, fp0, fp31 // " (fp0-1)
                tPIE2 += (tPIE1 + 1.0f) * 1.1f;
00000068: C0220000 lfs     fp1, @1361(RTOC) // " (fp1)
0000006C: C0020000 lfs     fp0, @1362(RTOC) // " (fp0)
00000070: EC00F82A fadds   fp0, fp0, fp31 // " (fp0)
00000074: EFC1F03A fmadds  fp30, fp1, fp0, fp30 // " (fp0-1)
                }
                }
00000078: 2C1E0000 cmpwi   r30, 0 // test count
0000007C: 3BDEFFFF subi   r30, r30, 1 // bump count
00000080: 4082FFC4 bne     *-60 // (loop till zero)
                ; $00000044
                RestorePowerPCVolatileFPRegs(&tFPURegsRec);
00000084: 38610038 addi   r3, SP, 56 // point r3 at tFPURegsRec on stack
00000088: 48000001 bl     .RestorePowerPCVolatileFPRegs // call our restore routine

0000008C: 80010098 lwz    r0, 152(SP) // return address -> r0
00000090: 38210090 addi   SP, SP, 144 // fix stack
00000094: CBE1FFF8 lfd   fp31, -8(SP) // restore volatile float regs (fp30-31)
00000098: CBC1FFF0 lfd   fp30, -16(SP)
0000009C: 7C0803A6 mtlrr r0 // move return address to link register
000000A0: 83E1FFEC lwz   r31, -20(SP) // restore volatile GP regs (r30-31)
000000A4: 83C1FFE8 lwz   r30, -24(SP)
000000A8: 4E800020 blr // return (branch to link register)

```

New (and Improved!) Glue Code:

Since the example routine only uses the volatile FP regs 0 and 1, the glue code can reduce the struct, save and restore to ONLY define, save and restore those registers. Here is the optimized solution:

```

#if TARGET_CPU_PPC
typedef struct FPUREgs
{
    float   FPR00;
    float   FPR01;
/*
    float   FPR02;
    float   FPR03;
    float   FPR04;
    float   FPR05;
    float   FPR06;
    float   FPR07;

```

```

float    FPR08;
float    FPR09;
float    FPR10;
float    FPR11;
float    FPR12;
float    FPR13;
*/
float    FPSCR;
} FPUREgsRec, *FPUREgsPtr, **FPUREgsHdl;

// function prototypes
asm void SavePowerPCVolatileFPRs(FPUREgsPtr pFPUREgsPtr);
asm void RestorePowerPCVolatileFPRs(FPUREgsPtr pFPUREgsPtr);

//
// Save the non-volatile PowerPC floating pointer registers
// into the passed in structure.
//
asm void SavePowerPCVolatileFPRs(FPUREgsPtr pFPUREgsPtr)
{
    stfd    fp0, FPUREgs. FPR00(r3)
    stfd    fp1, FPUREgs. FPR01(r3)
/*
    stfd    fp2, FPUREgs. FPR02(r3)
    stfd    fp3, FPUREgs. FPR03(r3)
    stfd    fp4, FPUREgs. FPR04(r3)
    stfd    fp5, FPUREgs. FPR05(r3)
    stfd    fp6, FPUREgs. FPR06(r3)
    stfd    fp7, FPUREgs. FPR07(r3)
    stfd    fp8, FPUREgs. FPR08(r3)
    stfd    fp9, FPUREgs. FPR09(r3)
    stfd    fp10, FPUREgs. FPR10(r3)
    stfd    fp11, FPUREgs. FPR11(r3)
    stfd    fp12, FPUREgs. FPR12(r3)
    stfd    fp13, FPUREgs. FPR13(r3)
*/
    mffs    fp0                                // copy fpscr into fp0
    stfd    fp0, FPUREgs. FPSCR(r3)           // save fpscr
    blr
}

//
// Restore the non-volatile PowerPC floating pointer registers
// from the passed in structure.
//
asm void RestorePowerPCVolatileFPRs(FPUREgsPtr pFPUREgsPtr)
{
    lfd     fp0, FPUREgs. FPSCR(r3)           // load saved fpscr into fp0
    mtfsf   0xff, fp0                        // move it back into the fpscr

    lfd     fp0, FPUREgs. FPR00(r3)
    lfd     fp1, FPUREgs. FPR01(r3)
/*
    lfd     fp2, FPUREgs. FPR02(r3)
    lfd     fp3, FPUREgs. FPR03(r3)
    lfd     fp4, FPUREgs. FPR04(r3)
    lfd     fp5, FPUREgs. FPR05(r3)
*/

```

```
    lfd      fp6, FPUREgs.FPR06(r3)
    lfd      fp7, FPUREgs.FPR07(r3)
    lfd      fp8, FPUREgs.FPR08(r3)
    lfd      fp9, FPUREgs.FPR09(r3)
    lfd      fp10, FPUREgs.FPR10(r3)
    lfd      fp11, FPUREgs.FPR11(r3)
    lfd      fp12, FPUREgs.FPR12(r3)
    lfd      fp13, FPUREgs.FPR13(r3)
*/
    blr
}
#endf // #if TARGET_CPU_PPC
```

Summary

- PowerPC floating point registers can be used at interrupt time as long as the developer remembers to save and restore all the non-volatile registers at the beginning and end of his interrupt code.

Further References

- [Inside Macintosh: PowerPC System Software](#), pp. 1-45.
- ["Mac OS Runtime Architectures," \(034-0206\)](#), pp. 4-8.

Downloadables



[Acrobat version of this Note \(49K\).](#)

Acknowledgments

Thanks to: Jim Murphy, Tim Carroll, and the infamous Quinn "The Eskimo!"

To contact us, please use the [Contact Us](#) page.
Updated: 15-March-99

[Technotes](#) | [Contents](#)
[Previous Technote](#)