New Technical Notes
Macintosh

Developer Support

FPU Operations on Macintosh Quadra Computers
Hardware **M.HW.QuadraFPU**

Revised by: Tim Dierks September 1992
Written by: Jon Okada, SANEitation Engineer, DTE June 1992

This Technical Note discusses floating-point unit (FPU) instruction support on Macintosh Quadra platforms with special emphasis given to compatibility and performance concerns.

Changes since June 1992: Added warning to check for an FPU before attempting to execute FPU instructions.

Introduction

The Macintosh Quadra computers are the first Apple products to use the Motorola 68040 microprocessor, which has an on-chip floating-point unit (FPU). This feature enables the Quadra to perform basic floating-point operations much faster than a Macintosh platform that employs an MC68881/2 floating-point coprocessor working in conjunction with an MC68020/030 microprocessor. This Note addresses compatibility and performance issues for Quadra computers executing FPU instructions either programmed explicitly in assembly language or generated by compilers (`-mc68881` and `-elems881` modes for MPW compilers).

While all currently available 68040 processors have an onboard FPU, it is important to use Gestalt to verify the existence of a floating-point coprocessor before attempting to use any FPU instructions. Motorola has announced a variant of the 68040 without an FPU unit; this chip has most of the caching characteristics of the current 68040, but does not support the 68881/2 opcode set.

Unfortunately, the FPU circuitry in the 68040 does not by itself support the full functionality of the MC68881/2. Motorola has provided a floating-point software package (FPSP) which emulates all of the MC68881/2 functionality that is not provided by the 68040. This package resides in the operating system of the Quadra. When the 68040 requires emulation services in the course of executing an FPU instruction, it traps to the FPSP via one of several exception vectors, depending on the type of emulation that is needed. The combination of the 68040 and FPSP enables Quadra computers to run old user code without modification unless the code uses floating-point exception handlers.

If user code includes floating-point exception handlers, the handlers must be modified to reflect the `FSAVE` state frames of the 68040, which differ from those of the MC68881/2. In addition, vectoring to such handlers for the 68040 must be done with care in order that entry to the FPSP not be compromised.

Whenever the 68040 in a Quadra invokes the FPSP, performance inevitably will suffer relative to an MC68881/2 platform because the software emulation of complex algorithms involving floating-point calculations and exception state simply cannot outperform dedicated hardware and microcode. In addition, the instruction cache must cope with many instructions of emulation code to accomplish what the MC68881/2 does in a single FPU instruction. Finally, FPSP intervention flushes the FPU pipeline, thus negating any performance enhancements achievable through overlapping execution of FPU instructions.

FPU Instructions Provided by the 68040

The following FPU instructions are supported by the 68040:

`FABS`	Absolute value
`FDABS`*	Absolute value rounded to double precision
`FSABS`*	Absolute value rounded to single precision
`FADD`	Addition
`FDADD`*	Addition rounded to double precision
`FSADD`*	Addition rounded to single precision
`FBcc`*	Branch on FP condition
`FCMP`	Comparison (sets FP condition codes)
`FDBcc`	Test FP condition, decrement D register, and branch
`FDIV`	Division
`FDDIV`*	Division rounded to double precision
`FSDIV`*	Division rounded to single precision
`FMOVE`	Move FP data or system control register
`FDMOVE`*	Move to FP data register rounded to double precision
`FSMOVE`*	Move to FP data register rounded to single precision
`FMOVEM`	Move multiple FP data registers
`FMUL`	Multiplication
`FDMUL`*	Multiplication rounded to double precision
`FSMUL`*	Multiplication rounded to single precision
`FNEG`	Negation
`FDNEG`*	Negation rounded to double precision
`FSNEG`*	Negation rounded to single precision
`FNOP`	No operation (flushes FPU pipelines and forces pending FP exceptions)
`FRESTORE`†	Restore internal FPU state saved by `FSAVE`
`FSAVE`†	Save internal FPU state
`FScc`	Set byte integer according to FP condition
`FSGLDIV`	Single precision division
`FSGLMUL`	Single precision multiply
`FSQRT`	Square root
`FDSQRT`*	Square root rounded to double precision
`FSSQRT`*	Square root rounded to single precision
`FSUB`	Subtraction
`FDSUB`*	Subtraction rounded to double precision
`FSSUB`*	Subtraction rounded to single precision

FTRAPcc	Trap on FP condition
FTST	Test FP operand and set FP condition codes

* Precision-constraining operation is not provided by MC68881/2; precision of instruction supersedes that set in the FP control register (FPCR).

† Privileged instruction.

Processing of these FPU instructions is usually handled entirely by the 68040. The FPSP is invoked if an unsupported data type or format is involved or if an exceptional condition is generated that requires fix-up of FPU state by emulation.

FPSP Overview

The FPSP provides three basic emulation services for the 68040. First, it emulates many MC68881/2 instructions, including all transcendental functions and some arithmetic instructions. Second, the FPSP handles instructions that involve certain data classes (unnormalized and denormal floating-point numbers) or the packed decimal data format, which are not supported by the 68040 hardware. Finally, the FPSP provides exception handlers for certain floating-point exception conditions in order to emulate MC68881/2 behavior when user traps are either disabled or enabled. In the latter case, after completing its exception processing, the FPSP passes control to the user-provided handler.

On Macintosh Quadra platforms executing MC68881/2 instructions, entry to the FPSP occurs automatically by trapping via one of several low-memory exception vectors, depending on which emulation service is required. The system installs the exception vector entries to the FPSP at boot time, and **applications should not tamper with these vectors.** Because the FPSP preempts the exception vectors for certain user-provided handlers in the MC68881/2 model, compatibility is a problem for old user code that contains floating-point exception handlers. Later sections will address the issues of compatibility in more detail.

Emulation of Unimplemented FPU Instructions

The following MC68881/2 arithmetic instructions are emulated by the FPSP, which produces results and exceptions identical to MC68881/2 platforms:

FGETEXP	Extract binary exponent of source
FGETMAN	Extract mantissa (significand) of source
FINT	Round source to integral value, using rounding mode in the FPCR
FINTRZ	Round source to integral value, using round-to-zero mode
FMOD	Modulo remainder of destination ÷ source with sign and lowest seven bits of quotient delivered in FP status register (FPSR) quotient byte
FMOVECR	Move constant ROM to FP data register
FREM	IEEE remainder of destination ÷ source with sign and lowest seven bits of quotient delivered in FPSR quotient byte
FSCALE	Scale (multiply) destination by 2^((int) source).

The following MC68881/2 transcendental functions are emulated by the FPSP:

FACOS	Inverse (arc) cosine (radians)
FASIN	Inverse (arc) sine (radians)
FATAN	Inverse (arc) tangent (radians)
FATANH	Inverse (arc) hyperbolic tangent
FCOS	Cosine of source in radians
FCOSH	Hyperbolic cosine
FETOX	Base e power (e^source)
FETOXM1	e^source - 1.0
FLOG10	Base 10 logarithm
FLOG2	Base 2 logarithm
FLOGN	Base e (natural) logarithm
FLOGNP1	Base e (natural) logarithm of (source + 1.0)
FSIN	Sine of source in radians
FSINCOS	Simultaneous sine and cosine (two destination registers)
FSINH	Hyperbolic sine
FTAN	Tangent of source in radians
FTANH	Hyperbolic tangent
FTENTOX	10.0^source
FTWOTOX	2.0^source

The algorithms used by the FPSP to calculate transcendental functions are both accurate and fast. Results will not always agree with those of the MC68881/2. When they disagree, the FPSP is generally more precise. The performance of the 68040 FPSP on transcendental functions is roughly equivalent to that of a similarly clocked MC68030/MC68882 combination.

When the 68040 in a Quadra attempts to execute any of the unimplemented MC68881/2 instructions, it traps, via vector number 11, the unimplemented F-Line opcode exception vector stored at vector offset (low-memory address) $002C to the FPSP. The corresponding exception handler in the FPSP saves the FPU state, decodes the instruction, fetches the operand(s), emulates the unimplemented instruction, and restores the appropriate state to the FPU. Operands involving unsupported data types or format are processed appropriately by this exception handler. To the user, the emulated instructions appear as atomic operations that produce valid results and that signal the proper floating-point exceptions. If an emulated instruction raises an enabled floating-point exception, program flow will vector to the appropriate user exception handler.

If the code executing in a Quadra contains an F-Line opcode that is undefined by the instruction sets of both the 68040 and MC68881/2, trapping to the FPSP via vector 11 also applies. In this case, the handler recognizes that no emulation is necessary, and it passes control to the system F-Line exception handler via a secondary vector stored in low memory.

Compatibility Note

If an application, such as a development or debugging environment, needs to install its own F-Line exception handler on Quadra platforms, it must **not** overwrite vector 11 at offset

$002C. If it does, emulation of the unimplemented MC68881/2 instructions will be lost with disastrous effects to the executing program. Instead, the secondary F-Line exception vector, located at address **$1FC8**, should be used on Quadra platforms. As is the case on MC68881/2 platforms, the application should save the inherited F-Line exception vector (secondary vector in the case of Quadra platforms) and restore it upon termination or context switch.

Unimplemented Data Type/Format Support in the FPSP

The FPU in the 68040 does not support all of the floating-point data types and formats of the MC68881/2. The following data types require FPSP support:

> denormalized single (S), double (D), or extended (X) precision operand to an FPU instruction; and unnormalized X operand to an FPU instruction.

The following data format requires FPSP support:

> packed decimal real (P) format as source or destination for an FPU instruction.

When the 68040 encounters an unimplemented data type or format in the course of executing a hardware-supported FPU instruction, it traps, via exception vector 55, the FP unimplemented data type exception vector stored at vector offset (low-memory address) $00DC to the FPSP. Prior to the release of the 68040, this address was unassigned but reserved by Motorola. The unimplemented data type exception handler in the FPSP takes the appropriate action for the instruction and the exceptional operand or format.

For denormal S, denormal D, and all P format source operands, the FPSP converts the values to the normalized X equivalents, restores FPU state, and restarts the operation. If a source operand is an unnormalized X that can be converted to a normalized X, the instruction is also completed as described. If the instruction is a move out to P format in memory (FMOVE.P FPn,<ea>), the FPSP emulates the conversion from the extended source format to P format and writes the result to the effective address.

For denormal X operands or unnormalized X operands that reduce to denormal X values, the FPSP converts such operands to an internal normalized format that contains an extra exponent bit, restores state to the FPU, and restarts the operation if no exponent wrap condition will occur (for example, division of a denormal value by another denormal value). Otherwise, the FPSP emulates the entire instruction.

Denormalized values resulting from instructions executed by the 68040 hardware do not generate the unimplemented data type exception. Instead, a non-maskable underflow exception occurs which invokes a handler in the FPSP. This handler rounds the internal result appropriately according to the specified rounding precision and direction and delivers the result.

In the case of instructions that are emulated by the FPSP, the processing of unimplemented data type/format operands is handled within the confines of the emulation process. That is, the 68040 traps to the FPSP's unimplemented instruction handler, which is capable of recognizing and dealing with such operands.

Instructions, whether emulated or not, that use the P format as either source or destination have relatively poor performance because they require emulation of binary-to-decimal or decimal-to-binary conversions.

Idiosyncrasies

Binary operations (source and destination operands are both inputs) with P format source operands should avoid using FP1 as the destination operand because a bug in the FPSP causes spurious results in this case. If an unimplemented data type or format occurs as input to an operation, the exception is posted by the 68040 when the next FPU instruction is attempted. This deferred exception handling may appear not to deliver the correct result in a debugging environment that installs a breakpoint prior to the second FPU instruction.

FPSP Exception Handlers

Certain floating-point exception conditions on the 68040 require intervention by the FPSP in order to fix up results or other state. Some of the FPSP exception handlers are non-maskable in the sense that they are executed regardless of whether or not the exception is trap-enabled by the user. All of the FPSP floating-point exception handlers, whether non-maskable or not, are vectored via Motorola-designated locations in low-memory supervisor address space. If a user-enabled exception occurs, the FPSP exception handler is executed first before vectoring occurs to the user handler via a secondary vector maintained by the Macintosh Quadra system. The user code must not modify the primary floating-point exception vectors to FPSP exception handlers. A later section will describe installation of user exception handlers.

The following is a brief description of FPSP exception handlers:

Branch/Set on Unordered (BSUN)

This maskable handler is invoked only if the user has enabled the BSUN exception. Entry to this handler is via vector number 48 stored at location `$00C0`. This handler updates the floating-point instruction address register (FPIAR) to contain the address of the floating-point branch/set instruction that generated the exception. It then invokes the user's handler via a secondary BSUN vector.

Inexact Result (INEX1/INEX2)

No FPSP handler is required. When enabled, INEX1 or INEX2 exceptions invoke the user's handler via vector number 49 at location `$00C4`.

Divide by Zero (DZ)

No FPSP handler is required. When enabled, the user's DZ handler is invoked via vector number 50 at location `$00C8`.

Underflow (UNFL)

This non-maskable handler is entered via vector number 51 at location `$00CC`. It determines and stores the properly rounded underflow result based upon the value of the intermediate

result and the rounding precision/direction modes stored in the FPCR. If underflow is enabled in the FPCR, the user's handler is invoked via a secondary UNFL vector.

Operand Error (OPERR)

This non-maskable handler is entered via vector number 52 at location `$00D0`. It provides compatibility of results with the MC68881/2 for B, W, and L destination formats when the source operand is a NaN (Not-a-Number), infinity, or value too large for the integer format. If the OPERR exception is user-enabled, the FPSP handler invokes the user's handler via a secondary OPERR vector.

Overflow (OVFL)

This non-maskable handler is entered via vector number 53 at location `$00D4`. It determines and stores the properly rounded overflow result based on the value of the intermediate result and the rounding modes stored in the FPCR. If overflow is enabled in the FPCR, the user's handler is invoked via a secondary OVFL vector.

Signaling Not-a-Number (SNAN)

This non-maskable handler is entered via vector number 54 at location `$00D8`. It provides compatibility of results with the MC68881/2 for B, W, and L destination formats. If the SNAN exception is user-enabled, program flow is directed to the user's handler via a secondary vector.

If a program enables no floating-point exceptions in the FPCR, compatibility is not an issue. In this case, no user exception handlers need be installed. The program traps to non-maskable FPSP handlers as required for any fix-up of exceptional results or FPU state and then resumes execution.

Performance degradation by non-maskable FPSP floating-point exception handling is minimal in most cases because such intervention is rarely needed. The most common exception, INEX2, requires no FPSP support. Underflows and overflows are infrequent when the default extended rounding precision is employed. OPERR occurrences are also rare, unless many out-of-range conversions occur from floating-point to integer formats.

User Floating-Point Exception Handlers

Users who require floating-point exception handlers in their applications running on Macintosh Quadra platforms must exercise some care in both the writing and the installation of such handlers. Moreover, if an application also targets Macintosh computers with MC68881/2 coprocessors and intends to resume processing via an `RTE` in an exception handler, its exception handlers must query which kind of FPU (MC68881/2 or 68040) is present and then execute hardware-specific code based on the query result. The reader is urged to consult the user manuals for the 68040 and MC68881/2 for details not covered by this Note.

Each floating-point exception on the 68040 is reported by either the conversion unit (CU) or normalization unit (NU) pipeline stage of the FPU. Exceptions reported by the CU are called E1 exceptions; they are detected relatively early in the execution of an FPU instruction. Exceptions reported by the NU are called E3 exceptions; they are detected late in the execution of FPU instructions as the NU attempts to normalize and round the intermediate result for storage in a destination FP register. E1 exceptions include all floating-point exception types.

The only E3 exceptions are OVFL, UNFL, and INEX2 occurring on opclass 0 (register-to-register) and opclass 2 (memory-to-register) instructions. If both E3 and E1 exceptions exist at the same time, the E3 exception should be handled first, allowing the 68040 to subsequently trap to handle the pending E1 exception.

There are two `FSAVE` stack frames for floating-point exceptions on the 68040. E1 exceptions produce the unimplemented instruction FPU state frame, and E3 exceptions produce the busy FPU state frame. Both of these frames begin with a 1-byte version number followed by a 1-byte frame length. The version number for Quadra 68040s is $41. For this version of the 68040, the frame length for E1 exceptions is $30, making the unimplemented instruction FPU state frame 52 bytes in size (counting the 4-byte header). The busy frame for E3 exceptions has a frame length of $60 and total size of 100 bytes.

Both 68040 floating-point exception `FSAVE` stack frames contain information that may be of use to the user's exception handler. There are two 12-byte fields containing the source and destination operands in extended precision. There are two 3-bit tag fields which classify the source and destination operands as to whether they are normalized, denormalized, zero, infinite, or NaN.There are two bits (E1 and E3) which, if set, indicate which pipeline stage of the FPU (CU or NU) detected the pending exception(s). Both `FSAVE` frames encode the command word of the exceptional floating-point instruction, albeit in different fields.

As a minimum, user floating-point exception handlers on 68040 platforms must issue an `FSAVE` instruction as the first FPU operation, clear the exception state of the FPU, and resume processing via the `RTE` instruction. For E3 exceptions, the E3 bit in the `FSAVE` stack frame must be cleared and the `FRESTORE` instruction must be issued prior to the `RTE` instruction. For E1 exceptions, the minimum requirement is to throw away the `FSAVE` stack frame and to resume processing via `RTE`. Another method of clearing the exception state for E1 exceptions is to clear the E1 bit in the `FSAVE` stack frame and issue the `FRESTORE` prior to the `RTE`. The E1 and E3 bits are bits 2 and 1 (bit position 0 representing the least significant bit), respectively, of the byte which is located 28 bytes below the high-address end of either `FSAVE` frame.

Minimum Floating-Point Exception Handler for the MC68881/2 and Quadra

The following code sequence serves as a minimum handler for all enabled floating-point exceptions except BSUN on both with MC68881/2 platforms and Quadra computers. This handler simply clears the exceptional condition in the FPU and resumes execution without attempting to modify any other FPU state. A minimal BSUN handler would require additional intervention (via one of four methods outlined in the user manuals for the 68040 and the MC68881/2) to prevent infinite looping on the BSUN trap.

```
; ********************************************************************
; Minimum user handler for enabled INEX, DZ, UNFL, OPERR, OVFL,
; or SNAN floating-point exception on either MC68881/2 or
; Macintosh Quadra platforms.
;
; NOTE:  For enabled DZ, OPERR, and SNAN exceptions for instructions
;        with FP register destinations, no result is delivered at all to the
;        destination register.
```

```
; ************************************************************************
HANDLER:
              FSAVE         -(SP)         ; save internal FPU state
              MOVE.L        D0,-(SP)      ; save D0, STACK:  D0 save < FSAVE frame
              MOVEQ         #0,D0         ; zero D0
              MOVE.B        4(SP),D0      ; NULL frame?

              BEQ.B         @NULL         ; yes, restore D0 and FPU state

              CMPI.B        #$41,D0       ; Quadra (68040) ID?

              BNE.B         @COPROC       ; no, assume MC68881/2

; Quadra FSAVE frame
              MOVE.B        5(SP),D0      ; D0 <- frame size

              BEQ.B         @NULL         ; restore state if 68040 IDLE frame

; Quadra UNIMPLEMENTED INSTRUCTION or BUSY FSAVE frame
              SUBI.B        #20,D0        ; D0 <- offset of E1/E3 byte from (SP)
              BCLR.B        #1,(SP,D0)    ; test and clear E3 byte
              BNE.B         @NULL         ; restore state if E3 was set

              BCLR.B        #2,(SP,D0)    ; E1 exception, clear E1 byte

; Restore state and resume execution
@NULL: MOVE.L        (SP)+,D0      ; restore D0, STACK:  FSAVE frame
              FRESTORE      (SP)+         ; restore FPU state
              RTE                         ; resume processing

; MC68881/2 IDLE FSAVE frame
@COPROC:      MOVE.B        5(SP),D0      ; D0 <- IDLE frame size
              ADDQ.B        #4,D0         ; compensate for D0 save value on stack
              BSET.B        #3,(SP,D0)    ; set bit 27 of BIU
              BRA.B         @NULL         ; restore state
```

Installation of User Floating-Point Exception Handlers

Current MPW language libraries (MPW 2.0.2 or later releases and Language Systems FORTRAN version 3.0) provide for the vectoring of user floating-point exception handlers in a consistent and portable fashion for both Quadra and MC68881/2 Macintosh platforms. The C functions `settrapvector` and `gettrapvector`, the Pascal procedures `SetTrapVector` and `GetTrapVector`, and the Language Systems FORTRAN subroutines `SetTrapVector` and `GetTrapVector` allow users to install and read vectors to their floating-point exception handlers via the use of the `TrapVector` structure. The relevant interface files for these operations are `{CIncludes}SANE.h`, `{PInterfaces}SANE.p`, and `{FIncludes}SANE.f`.

A `TrapVector` structure is composed of seven 4-byte fields that represent the entry-point addresses of the user's BSUN, INEX, DZ, UNFL, OPERR, OVFL, and SNAN exception handlers, respectively. `GetTrapVector` routines read the current floating-point exception vectors into a `TrapVector` structure. In order to install their own exception handlers, users

must first initialize a `TrapVector` structure with entry points of their handler routines and then invoke a `SetTrapVector` routine with that structure as the operand.

`GetTrapVector` and `SetTrapVector` routines involve privileged operations because they access Motorola low-memory vector table locations. For Quadra platforms, the situation is further complicated by the fact that five of the seven user floating-point exception vectors are stored by the system in secondary locations because the FPSP has preempted the original vector table locations. `GetTrapVector` and `SetTrapVector` implementations circumvent these difficulties by calling a system utility, **`PrivTrap`**, which does all of the work of querying or installing the user's vectors.

The `PrivTrap` Mechanism

`PrivTrap` is implemented as a system trap, **`$A097`**. Upon entry, it expects a selector value in register `D0.W` and a `TrapVector` structure address in address register `A0`. The `GetTrapVector` operation requires a selector value of 1; in this case, `PrivTrap` reads the current floating-point exception vectors into the `TrapVector` structure at `(A0)`. The selector value of 2 invokes the `SetTrapVector` operation; the user's exception vectors in the `TrapVector` structure at `(A0)` are installed appropriately in the system. In either case, registers `A0` and `A1` are modified upon exit.

As of the drafting of this Note, only the Quadra and PowerBook 170 platforms running system software version 7.0.1 have the `PrivTrap` mechanism built into their systems. Individual MPW library functions that require `PrivTrap` functionality first query if `PrivTrap` is installed. If it is not, the library routines will install and call a version of the trap appropriate for an MC68881/2 platform.

Implementation Notes

Since MultiFinder under system software version 6.0.x and Finder under current versions of System 7 do not include user exception vectors among the FPU state that is saved and restored at context switch, it is the responsibility of an application that enables floating-point exceptions to save inherited user exception vectors and to restore them upon termination or context switch. The inherited vectors may be read using the `GetTrapVector` operation. The application installs its floating-point exception handlers via the `SetTrapVector` operation. At context switch or program termination, `SetTrapVector` should be used to restore the appropriate exception vectors. If the above regimen is followed, the application's `TrapVector` structure may contain arbitrary values for vectors corresponding to disabled exceptions.

Performance Issues

In order to extract the maximum floating-point performance on a Macintosh Quadra, an application should avoid invoking emulation by the FPSP whenever possible. Unfortunately, FPU instruction sequences that optimize Quadra performance often degrade performance to some extent on MC68881/2 platforms. Programmers must always weigh the performance requirements of their various target platforms when writing floating-point code.

Transcendental Functions

Although all FPU transcendental function instructions are emulated by the FPSP on Quadra platforms, performance is comparable to a similarly clocked platform using the MC68882.

Unimplemented Arithmetic Functions

If deemed desirable for performance reasons on Quadra platforms, workarounds can readily be devised for most of the arithmetic FPU instructions that are emulated by the FPSP. The **FMOD** and **FREM** instructions are the notable exceptions since they involve an iterative algorithm in their most general cases. The functionality of the remaining unimplemented arithmetic instructions can be emulated as follows:

FGETEXP If the argument is a NaN or zero, return the argument. If the argument is infinite, return a NaN and signal OPERR. Otherwise, write the floating-point argument to stack, extract, and unbias the exponent using integer operations, and deliver the result to FPn using `FMOVE.L <ea>,FPn`.

FGETMAN If the argument is a NaN or zero, return the argument. If the argument is infinite, return a NaN and signal OPERR. Otherwise, write the floating-point argument to the stack in extended format, normalize the significand (mantissa) if necessary, set the exponent bits to $3FFF, retain the original sign bit, and deliver the result to FPn using `FMOVE.X <ea>,FPn`.

FINT If the argument is zero or if the exponent of the argument is greater than 62, return the argument. If the exponent of the argument is less than 31, round the argument to integral value by conversion to integer format via `FMOVE.L FPn,<ea>` followed by conversion back to X format via `FMOVE.L <ea>,FPm`. Otherwise, decompose the argument into an integral part (via integer operations on the X format on the stack) and a fractional part (via subtraction of the integral part from the argument), convert the fractional part to an integer via `FMOVE.L FPn,<ea>`, and add the integer to the integral part.

FINTRZ Using integer operations on the argument stored in extended format on the stack, test and zero out the fractional part. Set INEX2 if any fraction bits were nonzero. The test for inexactness may be omitted if the application is indifferent to INEX2 being signaled by this rounding operation.

FMOVECR Store desired constant in extended format in the code segment of program and load it via `FMOVE.X <ea>,FPn`.

FSCALE Convert the integral source operand n to a floating-point factor 2.0^n on the stack. Obtain the scale result via multiplication of that factor with the destination operand.

FINTRZ and Floating-Point → Integer Conversions

The most common compiler-generated unimplemented arithmetic FPU instruction is `FINTRZ` during conversions of floating-point values to various signed integer formats in C or FORTRAN source code. For example, to convert the value in `FPn` to 32-bit integer value at `<ea>`, a compiler will generate the following code sequence:

```
FINTRZ      FPn,FPm             ; truncate to integral value
FMOVE.L     FPm,<ea>            ; convert to integral format
```

If the application is running in (IEEE 754) default mode (FPCR = $00000000: no exceptions are enabled, rounding precision is extended, rounding direction is round-to-nearest), the following code sequence will accomplish the same conversion with optimal performance on a Quadra and with minimal performance degradation on an MC68881/2 platform:

```
FMOVE.L     #$00000010,FPCR     ; set round-to-zero mode
FMOVE.L     FPn,<ea>            ; truncate to integral format
FMOVE.L     #$00000000,FPCR     ; restore default modes
```

If the user's FPCR setting is not the default, the last sequence must be modified to save and restore the user's FPCR setting at the cost of several instructions and some temporary storage. Throughput for these conversions may be enhanced if the application requires an array of floating-point values to be converted, because the FPCR needs to be modified only once before and once after all conversions are done via the `FMOVE.L FPn,<ea>` step. Out-of-range source values result in degraded performance on Quadra computers due to nonmaskable vectoring to the OPERR handler in the FPSP.

Workarounds for conversions from floating-point values to the unsigned integer formats of C are more complicated and of necessity slower than those to signed integer formats.

Miscellaneous Performance Tips for Quadra Applications

In order to minimize trapping to the FPSP for handling of exceptional conditions, data types, or data formats, the following hints may prove useful:

- Applications should run with extended rounding precision set in the FPCR.
- Temporary storage for intermediate floating-point results should be in extended format and preferably in FP registers.
- Applications should avoid the generation of unnormalized extended format values via integer operations with subsequent reliance on the FPU to normalize the results.
- Applications should avoid the extensive use of the Motorola packed decimal (P) data format.

MPW QR6 Libraries

The MPW QR6 folder in the E.T.O. #6 Developers CD contains C and Pascal libraries that have been performance-tuned. In particular, some of the `-mc68881` mode implementations have been modified to obtain better performance on Quadra platforms. Included among the new implementations are conversions from floating-point to the unsigned integer formats of C. Unfortunately, conversions to signed integer formats are generated in-line by the C compiler and thus still include the `FINTRZ` instruction, which is emulated by the FPSP in Quadra platforms.

Summary

FPU operations on Macintosh Quadra platforms are performed by a combination of circuitry in the 68040 microprocessor and emulation code in the FPSP. The 68040 provides very fast implementations of most of the basic floating-point arithmetic functions in the MC68881/2 instruction set. The FPSP emulates all transcendental functions and some arithmetic functions. In addition, the FPSP handles instructions that involve certain data types/formats that are unsupported by the 68040 hardware and fixes up state when certain exceptional conditions arise during processing.

Compatibility of results relative to MC68881/2 platforms holds for all FPU arithmetic instructions, whether or not they are emulated on Quadra computers. Results for transcendental FPU instructions may differ, and they are generally more precise on the Quadra.

FPU applications that run with no floating-point exceptions enabled in the FPCR and that do not install an unimplemented F-Line Opcode handler will run without modification on both MC68881/2 and Quadra platforms. User unimplemented F-Line exception handlers are installed via vector 11 at address `$002C` on MC68881/2 platforms and via a secondary vector at address `$1FC8` on Quadra platforms. Similarly, installation of user floating-point exception handlers for enabled exceptions must take care not to overwrite entry points to the FPSP on Quadra platforms. MPW libraries provide high-level installation procedures for user floating-point exception handlers. If such handlers are to run on all FPU platforms, they must take into account the differences in `FSAVE` state frames for Quadra and MC68881/2 platforms.

Optimizing FPU performance on Quadra computers is largely a matter of understanding the conditions under which the FPSP is invoked and then avoiding such conditions via workarounds whenever possible. Code sequences thus optimized for Quadra computers will often provide less than optimal performance on MC68881/2 platforms.

Further Reference:

- *MC68881/MC68882 Floating-Point Coprocessor User's Manual*
- *MC68040 32-Bit Microprocessor User's Manual*
- *MC68040 Designer's Manual,* Section 3: Floating-Point Emulation
- *M68000 Family Programmer's Reference Manual*
- *IEEE Standard for Binary Floating-Point Arithmetic* (ANSI/IEEE Std 754-1985)