Porting Programs to PowerPC Numerics

This appendix contains information of interest to programmers who are porting programs from a non-Macintosh computer to run on a PowerPC processor-based Macintosh computer using PowerPC Numerics. If you are such a programmer and you think you are getting errors because of differences in numerics, you should read this appendix.

Porting applications to run in the PowerPC Numerics environment is easier than porting to other computers. Expressions that produce good results on other computers usually give at least as good results using PowerPC Numerics.

Note
If you are porting a program that uses SANE, read Appendix A, "SANE Versus PowerPC Numerics," instead of this appendix. ◆

Semantics of Arithmetic Evaluation

When you translate programs from one language to another, be aware of the hidden pitfalls in translation. For example, an operation in one language might have similar syntax to an operation in another language without being similar semantically. Here's an example of similar functions with different syntaxes:

- Fortran, `SIGN(A,B)` (two operands)
- BASIC, `SIGN(A)` (one operand)

Languages can also differ in how they treat mixed integers and reals. For example, Fortran truncates integer quotients to integers, so `3/7 = 0` (you have to write 3.0/7.0 to obtain a fraction). The programmer translating must be aware that the results of such expressions depend on the language used.

Languages also differ in how they convert from a real number to an integer. For example, in Fortran, assigning a floating-point value to an integer rounds toward zero.

Here are the operations used to truncate a real number to an integer in three languages:

- C: assignments and casts
- Fortran: `AINT`, `INT`
- Pascal: `Trunc`

Mixed Formats

On certain computers, the formats for single and double are identical except for their length. On those machines, for arguments passed by address, a calling routine can store data in one format and a called routine can read data in another format without apparent error.

If you have a program that exploits this confusion, you'll have to revise it before you can run it on a machine that uses PowerPC Numerics. (Type checking is of no help here; if the discrepancy was such that type checking could detect it, the original compiler would have caught it.)

Floating-Point Precision

Floating-point precision may differ from the original machine to the target machine.

Some computers have floating-point formats that have a wider range than the current PowerPC Numerics formats. Wider formats include the VAX™ H format, the IBM Q format, and the HP quad format. Programs use these wide formats for computation involving input data from a narrower format to minimize the occurrence of overflow and underflow and to preserve accuracy. The double-double data format provides enough precision to preserve accuracy; but it offers no greater range than the double format, so it will not protect against overflow and underflow. Keep in mind that problems may arise when a program uses formats wider than double-double.

CDC and Cray computers have a single format that is wider than IEEE single and a double format that is wider than IEEE double format. When porting code from those machines, you should consider changing type declarations from single to double format.

The Rules of Evaluation

Each computer uses different rules of evaluation. Here are three reasonable rules:

- Rule 1: Round the result to the wider of the two operand formats.
- Rule 2: Round the result to the widest available format.
- Rule 3: Round the result to the widest format in the expression.

Rule 1 is instant rounding. It is the rule on computers having many registers the same width as memory. This rule has been used by IBM and CDC Fortran since 1963. It is not part of the Fortran standard, though it is often thought to be.

Rule 2 is what SANE does by evaluating in extended precision. Other machines using this approach include the PDP-11C (using double precision) and floating-point coprocessors such as the 8087 and the MC68881. This approach does not take best advantage of machines with separate processing units for each floating-point format.

Rule 3 is what PowerPC Numerics does and is the way you do it when computing by hand. It was the rule in Fortran until 1963. By this rule, if you see an expression with mixed precision, you assume the user wants the widest visible precision.

With PowerPC Numerics, you can write code to simulate any of these rules. To simulate rule 1, use separate assignments when computing subexpressions. To simulate rule 2, convert all operands to double-double format before performing an expression.

For transported code, either you have to understand the programmer's tricks or you have to mimic the way rounding works on the programmer's machine. With PowerPC Numerics, you can set the rounding direction to mimic other machines.

The Invalid Exception

Many computers used to stop on an invalid operation, such as 0/0. Programmers have made the best of this and not bothered to test in advance for values that could cause an invalid operation. It is better to stop than to give a plausible but incorrect answer.

When a program written that way runs on PowerPC Numerics, it produces a NaN where it formerly would have stopped. The NaN might cause the program to take an unplanned branch and thus produce an erroneous answer. Because the program does not test for invalid operations, the user will not know whether the answers the program finally delivers have been influenced by exceptional events that formerly would have stopped the computer.

Programs sometimes contain code that depends on an ill-documented effect or on one that varies from machine to machine. If you have inherited such a program and you do not know what it does about exceptional conditions, here are some possible strategies:

- Insert tests on operands that could cause invalid operations.
- Change the program to make sure that NaNs propagate as NaNs rather than as plausible answers.
- After evaluations, add code to test the invalid flag and deliver a meaningful result or message and then clear the flag.

If you have a program with code you can't change and you distrust the results it gives when invalid operations occur, you should set up tests that halt programming on those invalid operations and set the environment to simulate the environment in which the program was designed to run.