This chapter describes the ways in which an expression with floating-point operations can be evaluated in the PowerPC Numerics environment. The environment does not require that all floating-point operations be performed with a certain precision. Instead, it lets each implementation choose the most efficient precision to use. An implementation can dictate that all floating-point operations be performed with a given precision, or an implementation may define a method by which the best possible precision is chosen for each expression. This chapter describes the two methods that numeric implementations can use to choose a precision and compares the methods using several examples.

You should read this chapter to learn how PowerPC Numerics determines the precision of a floating-point expression.

# About Expression Evaluation



The **evaluation format** of a floating-point operation is the data format used to evaluate the operation. An **expression evaluation method** is the means by which evaluation formats are determined. The IEEE standard does not cover expression evaluation methods, but the FPCE technical report does. Expression evaluation methods in PowerPC Numerics comply with the FPCE recommendations.

All PowerPC Numerics expression evaluation methods have a predefined minimum evaluation format, and they may or may not have widest-need evaluation. The **minimum evaluation format** is the narrowest evaluation format allowed for any operation. Any of the three floating-point data formats (single, double, or double-double) can be designated as the minimum evaluation format. **Widest-need evaluation** is a method used to determine the evaluation format for **complex expressions** (expressions with more than one floating-point operation). The following sections describe how expressions are evaluated without widest-need evaluation and with widest-need evaluation.

# Evaluating Expressions Without Widest Need

Without widest-need evaluation, a complex expression is considered as a series of **simple expressions** (expressions with only one floating-point operation), and the evaluation format of each simple expression is determined separately. The evaluation format of a simple expression is either its **semantic type** (the widest format used for its operands) or the minimum evaluation format, whichever is wider. For example, consider the operation

*s* * *d*

where *s* is a single-format variable and *d* is a double-format variable. The operation's semantic type is double because double is the widest format used for an operand. If the minimum format is defined to be single, the operation is evaluated in double precision

because double is wider than single. If the minimum format is double-double, double-double precision is used because double-double is wider than double. Evaluating this operation in double-double precision means that the values of both variables will be converted to double-double format before the multiplication is performed and that double-double format will be used for temporary storage of the result.

This expression evaluation method applies only to floating-point operations subject to the **usual arithmetic conversions** (automatic conversions performed in the C programming language). The following operations are subject to the usual arithmetic conversions:

- arithmetic operations
- comparison operations

The following operations are *not* subject to the usual arithmetic conversions:

- assignment
- assignment of actual function arguments to formal function parameters
- explicit conversions to different data types (for example, casts in C)

For example, consider the C expression

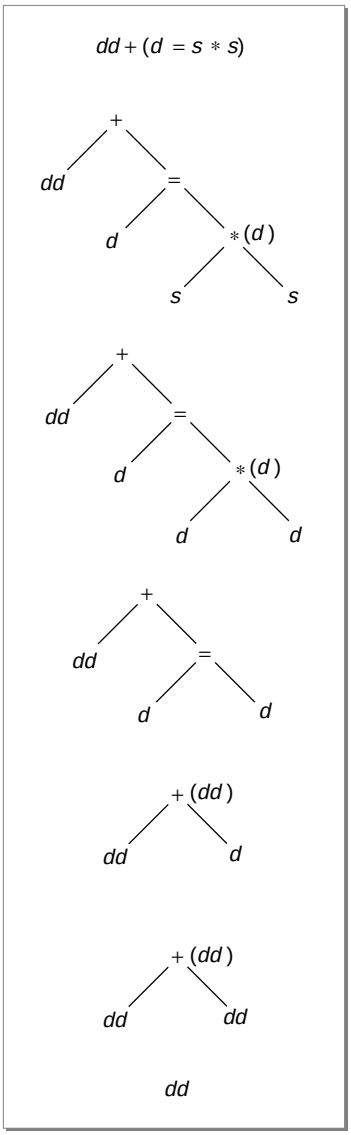
```
dd + (d = s * s)
```

where *dd* denotes a double-double format variable or number, *d* is double format, *s* is single format, and the minimum evaluation format is double. Without widest-need evaluation, this expression is treated as three simple expressions:

- *s* * *s*
- *d* assigned the result of *s* * *s*
- *dd* + the result of *d* = *s* * *s*

The semantic type of the first simple expression (*s* * *s*) is single, which is narrower than the minimum evaluation format, so it will be evaluated in double. The values of both of its operands are converted to double format and are then multiplied to produce a double result. The next simple expression is an assignment operation, which is not subject to the usual arithmetic conversions so the expression evaluation method does not apply. It produces a double format result also. Then, the last simple expression is considered. Its semantic type is double-double, so that will be the evaluation format. The result of the assignment is converted to double-double format, then added to the double-double variable. Figure 3-1 illustrates this process.

**Figure 3-1** Evaluating complex expressions without widest need




# Evaluating Expressions With Widest Need

Widest-need evaluation first looks at all of the operands of all of the subexpressions in a complex expression to determine the semantic type of the complex expression. As before, if the semantic type is wider than the minimum evaluation format, the semantic type is the evaluation format. If not, the minimum evaluation format is used. Only subexpressions with operations subject to the usual arithmetic conversions are considered when determining the evaluation format; operations such as assignment statements or casts are ignored.

After the evaluation format is determined, widest-need evaluation applies this format to the operands of the outermost operation in the expression using one of the following rules:

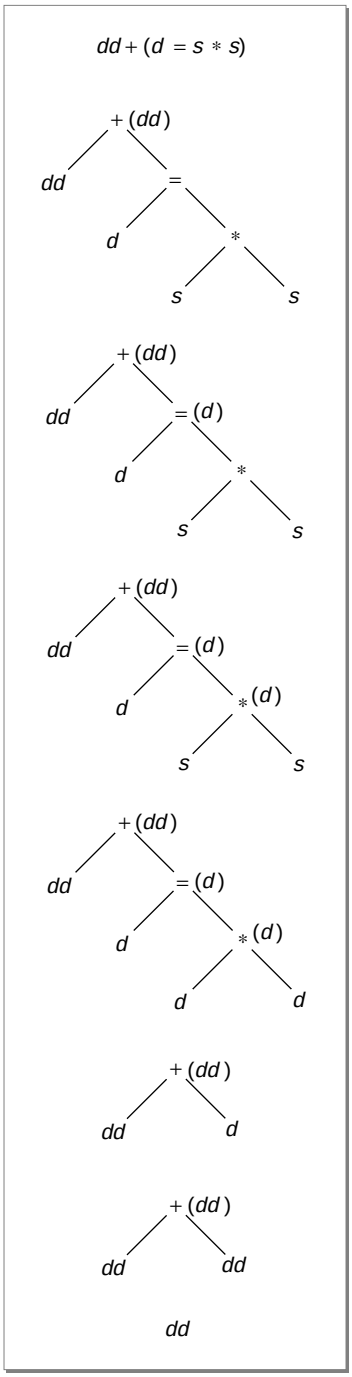
- If the operand is a floating-point variable or constant, it is converted to the evaluation format.
- If the operand is an operation subject to the usual arithmetic conversions (for example, arithmetic operations and comparison operations) or the assignment of values to function parameters, its operands are converted to the evaluation format before the operation is performed.
- If the operand is an operation not subject to the usual arithmetic conversions (for example, an assignment operation, function call, or cast), its evaluation format is determined separately from the outer expression. After the operation has been performed, its result is converted to the evaluation format of the complex expression.

These three rules are applied repeatedly until the end of the expression is reached. For example, consider the C expression in Figure 3-2. Widest-need evaluation looks at this expression as the addition of a double-double variable to the result of another expression. To determine the evaluation format of this addition operation, widest need first looks at all of the variables and constants in the entire expression that are not part of a function call, cast, or assignment operation. There is only one variable that meets these requirements, and it is in double-double format. Therefore, double-double format is the evaluation format of the addition operation.

Now, widest-need evaluation can apply the addition operation's evaluation format to the rest of the expression using the three rules just given. Addition is an operation subject to the usual arithmetic conversions, and so its operands will be converted before the operation is performed. The first operand is a double-double variable, so it will be converted to the evaluation format immediately. (In this case, the variable already is in the evaluation format.) The second operand is an assignment operation. The assignment operation is not subject to arithmetic conversions, so it will be performed before any conversion takes place. This means that the evaluation format for the assignment operation must be determined. The operation's semantic type is double, so it will be performed in double precision.

As before, this double format must now be applied to the operands of the assignment. The first operand is already in double format. The second operand is a multiplication operation. Because multiplication is subject to the usual arithmetic conversions, its operands are converted before the operation is performed. Both of the multiplication operation's operands are single-format variables, so the values of these two variables are converted to double. The multiplication operation is calculated in double precision. Now the assignment can be performed, resulting in a double-format number. This result of the assignment statement is now the second operand of the addition operation. It is converted to double-double format, and then the addition is performed in double-double precision.

**Figure 3-2** Evaluating complex expressions with widest need



# Comparisons of Expression Evaluation Methods

You can think of the difference between using and not using widest-need evaluation as the way these two expression evaluation methods navigate the parse tree for a complex expression. Widest-need evaluation determines the evaluation format of the topmost expression first and enforces that format on all lower expressions. If a complex expression is evaluated without widest need, the evaluation format of the bottommost expression is determined first, and the results are converted to wider formats as wider formats are encountered working back up the tree.

Figure 3-3 shows how an expression is evaluated using both methods. In this example, *dd* is a double-double format constant or variable, *d* is double format, and *s* is single format. The minimum evaluation format is single. This expression makes a call to a function named `dfunc`, which takes a parameter of type `double` and returns a double value.

If this expression is evaluated without widest need, the evaluation format of the multiplication operation (*s* `*` *s*) is determined first without regard to the rest of the expression. Its semantic type is single, which is the same as the minimum evaluation format, so it is evaluated in single precision. Its result is then converted to double precision when it is passed to the function `dfunc`, which takes a double parameter. The function returns a double result. The next expression is the addition operation, which has a semantic type of double-double. The addition will be performed in double-double precision because double-double format is wider than the minimum evaluation format. The double-format return value from `dfunc` is converted to double-double, the addition is performed, and a double-double result is returned.

If this expression is evaluated with widest-need evaluation, the evaluation format of the addition operation is determined first. All of the variables in the expression that are not assigned to function parameters or not part of an assignment statement or cast are looked at to determine the evaluation format. In this expression, the two variables considered are the *dd* variable and the `dfunc` function call. Because *dd* is double-double format, the evaluation format of the addition operation is double-double. Now, the double-double format is applied down the parse tree to the operands of the addition operation. The first operand is already in double-double format. The second operand is a function call. As explained on page 3-6, function calls are not subject to the usual arithmetic conversions, so their evaluation formats are determined independently of the outer expression and their results are determined before any conversion takes place. The evaluation format for the assignment of values to the parameters of `dfunc` is double because `dfunc` takes a parameter of type `double`. The multiplication operation is an operand to this operation, so the multiplication is performed in double precision. The result of `dfunc` is returned in double format, then is converted to double-double format before the addition is performed.

**Figure 3-3** Evaluating an expression with a function call

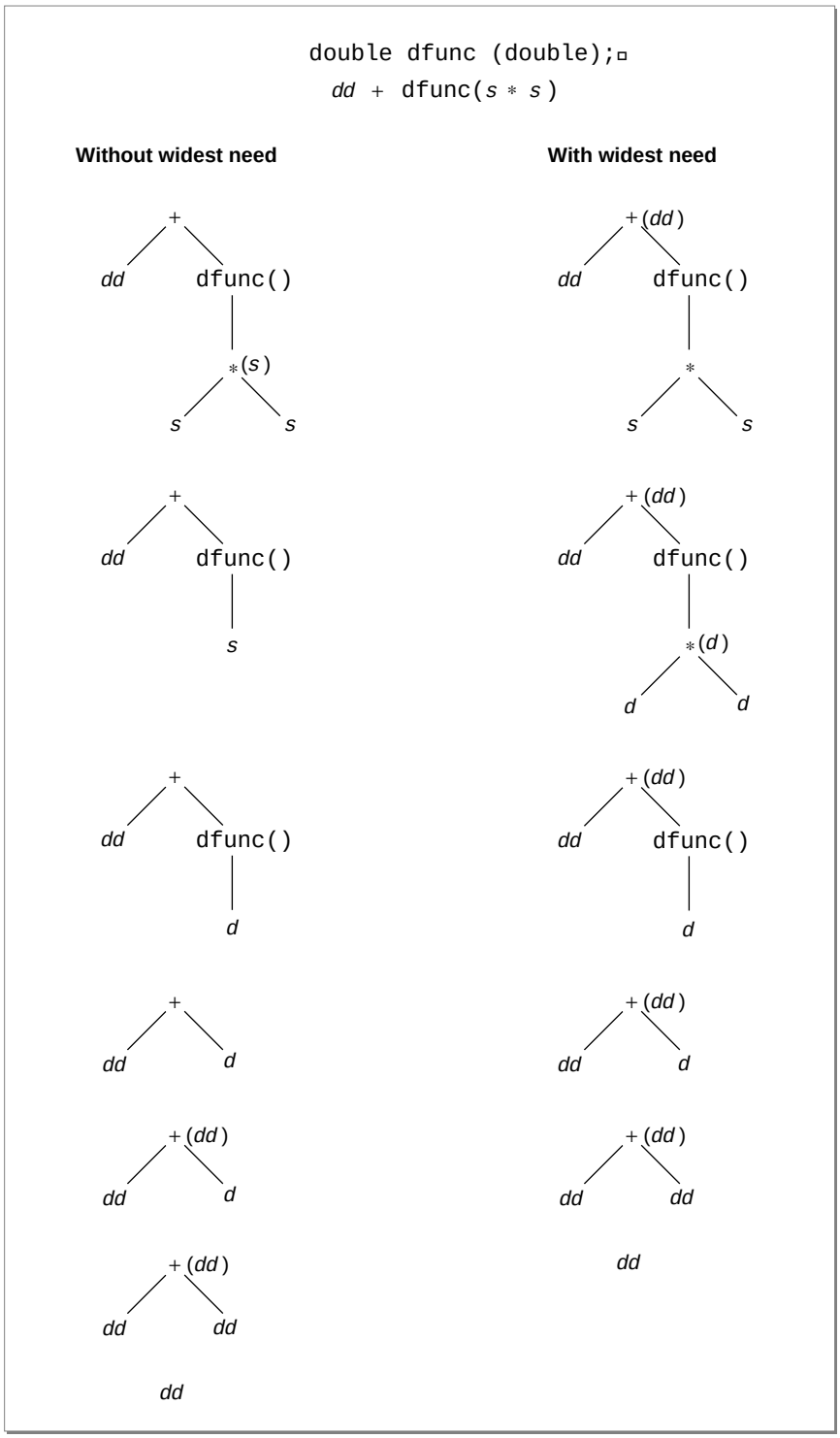
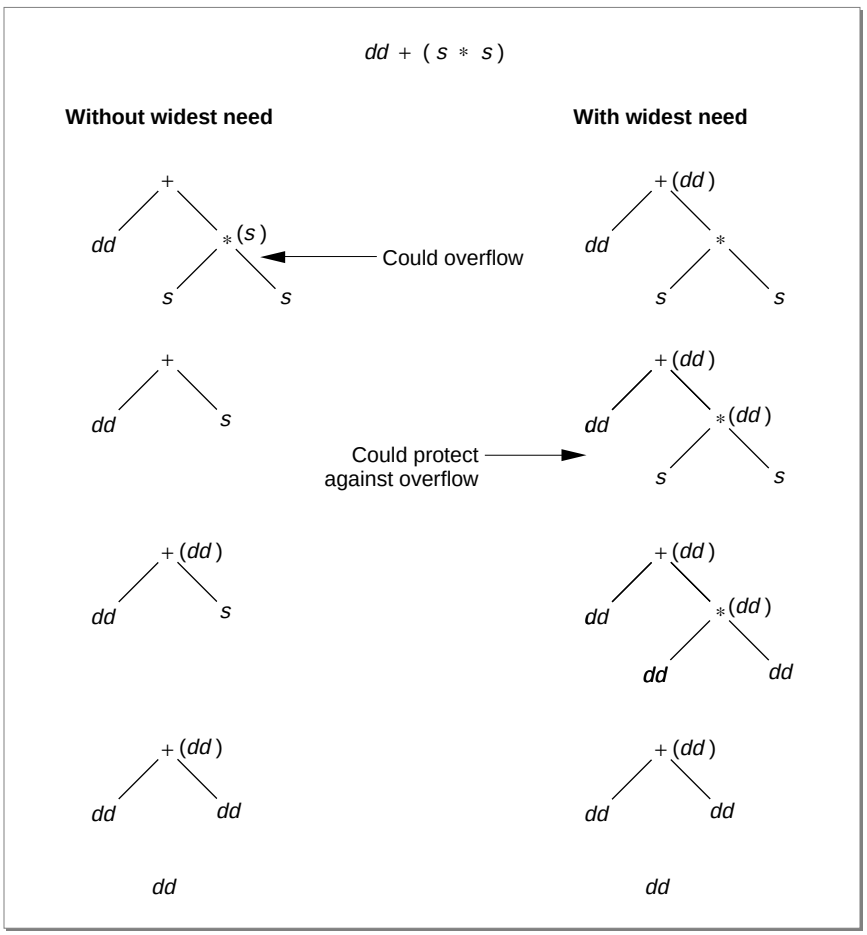


Figure 3-4 shows how widest-need evaluation protects against midexpression overflow and underflow better than expression evaluation methods that do not use widest need. In this example, *s* denotes a single-format variable or number, *d* is double format, *dd* is double-double format, and the minimum evaluation format is single.

**Figure 3-4** Evaluating an expression with arithmetic operations




Without widest-need evaluation, the expression in Figure 3-4 is considered as two separate simple expressions. The multiplication operation (*s* * *s*) is considered first. Its semantic type (single format) is the same as the minimum evaluation format, so the multiplication is performed in single precision. The semantic type of the addition operation is double-double, which is wider than the single minimum format. The addition operation is evaluated in double-double precision, so the value of its single-format operand is converted to double-double format before the result is calculated.

With widest-need evaluation, all of the operands in the complex expression are looked at first to determine the semantic type. The semantic type is double-double because of the double-double variable. This means that the multiplication of the two single-format variables is performed in double-double precision.

Suppose that the two single variables have the values $10^{38}$ and 10, respectively. Multiplying these two values produces $10^{39}$. However, $10^{39}$ is out of the range of single format. If these numbers are multiplied in single precision (that is, if widest-need evaluation is not used), it will produce $+\infty$ and a floating-point overflow exception. If the multiplication is evaluated in double-double precision (that is, if widest-need evaluation is used), the correct result is returned because $10^{39}$ is within the range of the double-double format.