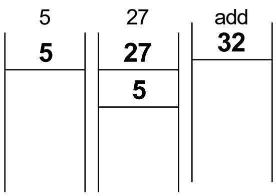
add and **sub**

Monad operators look to the stack for the numbers they need, that is, for their *operands*. The operator generally removes its operands from the stack and replaces them with whatever results that operator produces.

For example, the **add** operator causes Monad to remove the top two numbers from the stack, add them, and leave the sum on the stack. Thus, the program line below would affect the stack as illustrated at left.



add

    5 27 add

The *5* and the *27* are pushed onto the stack and the **add** operator then replaces them with their sum.

sub

The Monad **sub** operator works in a similar manner, with the program line

    8.3 6.6 sub

having the results diagrammed at left. The numbers *8.3* and *6.6* are pushed on the stack; the **sub** operator subtracts the top number on the stack from the number below it.

Stack Notation

The contents of the Monad stack is typically depicted in print as a line of numbers (or other data) with the top of the stack at right. Thus, a stack with *6* on top, *143.9* below it, and *–800* below that is printed:

    –800 143.9 6

Notice that this displays the numbers in the order in which they were originally placed on the stack.

Similarly, the effects of an operator on the stack may be in-dicated by showing the stack's initial condition (before the operator is executed), the operator's name, and then the contents of the stack after the operator was executed. Using this method, a demonstration of the effects of **add** could be expressed:

5 27 **add** ⇒ 32

.

Other Arithmetic Operators

Besides **add** and **sub**, Monad possesses a full range of arithmetic operators, including:

div Divide the second number on the stack by the top number on the stack. For example,

13 8 **div** ⇒ 1.625

idiv Divide the second number on the stack by the top number on the stack; only the integral part of the quotient is retained.

25 3 **idiv** ⇒ 8

mod Divide the second number by the top. In this case, only the remainder of the division is kept.

12 10 **mod** ⇒ 2

The operands passed to the **mod** and **idiv** operators must be integers.

mul Multiply the top two numbers on the stack, pushing the product onto the stack.

6 8 **mul** ⇒ 48

neg Reverse the sign of the number on top of the stack.

–27 **neg** ⇒ 27

These are the arithmetic operators we shall be using the most in this tutorial. Monad arithmetic operators, include **sqrt**, **exp**, **ceiling**, and **sin**.

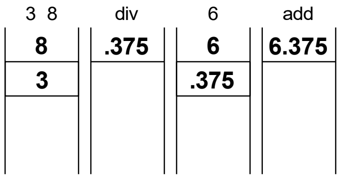
More-Complex Arithmetic

The use of a stack in Monad allows some freedom in exactly how an arithmetic process is carried out. For example, let us say that we wanted to calculate

$$6 + (3 \div 8)$$

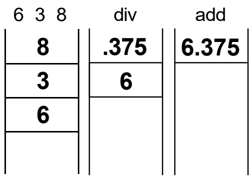
in Monad. Either of the following two program lines would leave the appropriate number on the stack.

- 3 8 div 6 add
- 6 3 8 div add



6+3/8, Example 1

In the first case (see illustration), we put *3* and *8* on the stack, divide the former by the latter, put *6* on the stack, and add it to the quotient below it.



6+3/8, Example 2

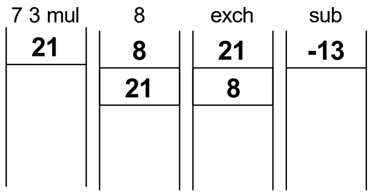
In the second case, the same operations are performed, but now we start out by putting all three of the numbers on the stack. Then we call the **div** operator, which divides the second number (*3*) by the top (*8*) and add the top two numbers (*6* and *.375*).

Similarly, the equation

$$8 - (7 \times 3)$$

can be expressed in at least two ways:

- 8 7 3 mul sub
- 7 3 mul 8 exch sub



8-7∗3

The second method introduces a new operator: **exch**. This operator exchanges the top two items on the stack. Note that in this example, the phrase *7 3 mul* places the two numbers on the stack and multiplies them, leaving the product, *21*, on the top of the stack. The number *8* is then pushed onto the stack, but this leaves the stack contents in the wrong order for our subtraction. The **sub** operator subtracts the top number from the second, which in this case would be *21* minus *8*, the opposite of what we want. The **exch** operator invoked at this point reverses the order of the top two numbers of the stack, putting them in the correct order for our subtraction.

Stack Operators

The **exch** operator is our first example of a *stack operator*, an operator whose function is to add, remove, or rearrange items on the Monad stack. There are several such operators, includ-ing:

clear Removes all items from the stack.

6 8 12 **clear** ⇒ —

dup Duplicates the top item on the stack.

6 **dup** ⇒ 6 6

pop Remove the the top element from the stack.

17 8 **pop** ⇒ 17

roll Roll stack contents. Take two numbers from the stack. The top number tells Monad how many times and in which direction to rotate the stack; the second number is how many items are to be rotated.

7 8 9 3 1 **roll** ⇒ 9 7 8

7 8 9 3 –1 **roll** ⇒ 8 9 7

2.1 OPERATOR SUMMARY

Stack Operators

clear $ob_1 ... ob_i \Rightarrow$ —
Remove all stack contents

dup ob ⇒ ob ob
Duplicate top of stack

exch $ob_1\ ob_2 \Rightarrow ob_2\ ob_1$
Reverse order of top two objects on stack

pop $ob_1\ ob_2 \Rightarrow ob_1$
Remove top of stack

roll $ob_{n-1} ... ob_0\ n\ j \Rightarrow ob_{(j-1) \bmod n} ... ob_0\ ob_{n-1} ... ob_{j \bmod n}$
Rotate *n* elements *j* times

Math Operators

add $n_1\ n_2 \Rightarrow n_1{+}n_2$
Add two numbers

div $n_1\ n_2 \Rightarrow n_1{\div}n_2$
Divide two numbers

idiv $n_1\ n_2 \Rightarrow \mathrm{int}(n_1{\div}n_2)$
Integer divide

mod $n_1\ n_2 \Rightarrow (n_1 \text{ MOD } n_2)$
Modulus

mul $n_1\ n_2 \Rightarrow n_1{\times}n_2$
Multiply two numbers

sub $n_1\ n_2 \Rightarrow n_1{-}n_2$
Subtract two numbers

from $n_1\ n_2 \Rightarrow n_2{-}n_1$
Subtract two numbers