An AUGMENT Interface for Brent's Multiple Precision Arithmetic Package

RICHARD P. BRENT
Australian National University
JUDITH A. HOOPER
University of Wisconsin–Madison
and
J. MICHAEL YOHE
University of Wisconsin–Eau Claire

The procedure required to interface Brent's multiple precision package MP with the AUGMENT precompiler for Fortran is described A method of using the multiple precision arithmetic package in conjunction with AUGMENT is discussed.

Key Words and Phrases arithmetic, multiple precision, extended precision, floating point, portable software, software package, precompiler interface, AUGMENT interface
CR Categories: 4.49, 5.11, 5.12

1. INTRODUCTION

The purpose of this paper is to demonstrate the ease with which a well-designed nonstandard arithmetic package may be interfaced with the AUGMENT precompiler for Fortran [6, 7]. We outline an interface and user instructions to enable one to use the Fortran multiple precision arithmetic package MP [1–4] in conjunction with AUGMENT. This makes the use of MP far more natural and convenient than its use without AUGMENT. With the aid of AUGMENT, the user declares multiple precision variables as type MULTIPLE, and then, for the most part, simply writes the program as though MULTIPLE were a standard Fortran data type.

2. WRITING THE INTERFACE

We assume that the reader is familiar with the AUGMENT precompiler, at least to the extent of knowing what is meant by such terms as *supporting package* and *description deck*. This degree of familiarity may be obtained by consulting [6].

This research was supported by the U.S. Army under Contract DAAG29-75-C-0024.
Authors' addresses R P Brent, Department of Computer Science, Australian National University, Canberra ACT 2600, Australia; J A. Hooper, Mathematics Research Center, University of Wisconsin-Madison, Madison, WI 53706, J.M. Yohe, Academic Computing Services, University of Wisconsin-Eau Claire, Eau Claire, WI 54701.

The supporting package to be interfaced with AUGMENT is the Fortran multiple precision arithmetic package described in [1]. This is a collection of portable subroutines which performs not only basic arithmetic operations in multiple precision, but also all of the ANSI standard mathematical functions and many nonstandard ones. The precision of the package is governed by the user at run time and may be changed during the course of a computation.

In interfacing this or any package with AUGMENT, we must specify the amount of storage to be allocated to each variable. This will place an upper limit on the operating precision of the multiple precision arithmetic package, although nothing prevents one from using a lower precision in computations. Increasing the precision beyond that provided in this standard interface is not difficult.

The first step in interfacing the package was to prepare the AUGMENT description deck. Fortunately, most of the multiple precision routines were in the form of subroutines, with the numbers of arguments expected by AUGMENT (and in the order expected); nearly all of the manipulations required for a complete package were already provided (in the form assumed by AUGMENT), and all of the subroutines in the package bore the prefix MP in their names. This made preparation of the interface extremely straightforward.

To prepare the description deck, we simply went down the list of routines in the multiple precision arithmetic package, associating them when possible with standard Fortran operations and functions. When such a natural association was not possible, we assigned function names (usually obtained by dropping the prefix MP from the routine name). The description of each routine was coded as per the instructions in [7]. In only a few cases were we unable to do this: most of the input/output routines and error checking routines could not be interfaced with AUGMENT (they must be called explicitly), and the routines which provide constants needed special attention, as described below. Routines which did not conform to the usual expectations of AUGMENT, such as the routine to add the quotient of two integers to a multiple precision number, were simply described as functions.

The routines used to generate constants posed a problem: AUGMENT assumes that routines will have at least one argument in addition to the result, and these routines did not. We therefore decided to write a new routine which "converts" the Hollerith name of the desired constant to the value of the constant. Once this routine was written, it seemed logical to include the capability of run-time conversion of Hollerith strings representing numeric constants.

We also wrote six trivial logical functions to allow AUGMENT to deal with the six logical operators in the context of multiple precision variables and some other routines to allow the user to inspect and modify the base, number of digits, sign, exponent, and digits of multiple precision numbers without needing to know the details of the implementation of the package. Finally, we added some input/output routines which are simpler to use with the AUGMENT interface than those originally included in the multiple precision arithmetic package. All of these routines were extremely straightforward to write and required a total of about 120 executable statements. A listing is given in [5, Appendix C].

The entire interface was written in less than half a day; the most time-consuming task was revising the documentation for the multiple precision pack-

age! The new MP routines have been tested on Univac 1100, IBM 370, CDC Cyber 76, and DEC-10 machines.

3. USE OF THE PACKAGE VIA AUGMENT

As explained in [5], the use of a nonstandard arithmetic package via AUGMENT is extremely simple. The majority of the package modules are invoked automatically by AUGMENT, the exceptions being mainly the input/output and error handling routines.

To use the package via AUGMENT, the user declares all multiple precision variables using statements of the form

```
MULTIPLE X, Y(10), Z
```

or

```
IMPLICIT MULTIPLE (A - H, O - Z)
```

(AUGMENT accepts type declarations via IMPLICIT statements, whether or not the Fortran compiler does; this is convenient when converting a program to multiple precision.) The majority of the program is then written just as though MULTIPLE were a standard Fortran data type.

The interface allows the use of all ANSI standard operators and functions with variables of type MULTIPLE as well as REAL, e.g., +, −, *, /, **, SIN, ATAN, etc. Some additional functions and environment parameters are provided, e.g., the error function, Bessel functions, machine precision, etc.

Constants may be introduced into the program by statements of the following types:

```
PI = 'PI'
X = '.1$'
```

The dollar sign on the second Hollerith literal is a sentinel to let the Hollerith-unpacking routine know when it has reached the end of the literal. The Hollerith-unpacking routine depends on the number of characters per word and is not portable but should be easy to adapt for most machines.

The user must set the various parameters for the package, as explained in [1] and [3]. Care must be exercised to ensure that the dimensions of the multiple precision variables communicated to the package are no greater than those used by AUGMENT in assigning space to the variables. One way of setting these parameters to default values is to include the statement

```
INITIALIZE MP
```

after the type declarations in the user's main program. This causes AUGMENT to generate a call on the routine MPINIT, which then sets the parameters to values fixed in the MPINIT subroutine. This is a bit of a cludge, but it works, provided the default values are what one really wants. Other ways of setting the package parameters are described in [3] and [5].

Once the program has been written, the following run stream will invoke

AUGMENT and cause the translated program to be written on a Fortran logical unit:

```
(invoke AUGMENT)
(Description Deck)
*BEGIN
(Source Program)
*END
```

The resulting program would then be compiled just like any other Fortran program, linked with the multiple precision library routines, and executed.

A complete list of the operations and functions available in the multiple precision arithmetic package, together with the manner in which they are invoked via AUGMENT, is given in [5, Appendix A]. An example program illustrating the use of the AUGMENT interface may be found in [3, Sec. 4.3].

4. CONCLUSION

We have demonstrated the method of interfacing a supporting package with the AUGMENT precompiler in the most convincing way possible: by actually doing it. Since the package is now much easier to use, the relatively small amount of work required to program the AUGMENT interface was well worthwhile.

A revised version of the multiple precision arithmetic package, incorporating the AUGMENT interface routines and description deck (but not AUGMENT), is available from the first author.

REFERENCES

1. BRENT, R P A Fortran multiple-precision arithmetic package. *ACM Trans. Math. Softw. 4*, 1 (March 1978), 57–70.
2. BRENT, R.P Algorithm 524. MP, a Fortran multiple-precision arithmetic package. *ACM Trans Math Softw 4*, 1 (March 1978), 71–81.
3. BRENT, R.P. MP users guide. Tech Rep. 54, Computer Center, Australian National Univ., Canberra, Australia, Sept. 1976 (revised July 1978).
4. BRENT, R.P. Remark on algorithm 524. MP, a Fortran multiple-precision arithmetic package. *ACM Trans. Math. Softw. 5*, 4 (Dec 1979), 518–519.
5. BRENT, R.P., HOOPER, J.A , AND YOHE, J.M. An AUGMENT interface for Brent's multiple precision arithmetic package Tech. Summary Rep. 1868, Mathematics Res. Cent., Univ. Wisconsin–Madison, Madison, Wis , Aug. 1978.
6. CRARY, F D. A versatile precompiler for nonstandard arithmetics. *ACM Trans. Math. Softw. 5*, 2 (June 1979), 204–217.
7. CRARY, F.D. The AUGMENT precompiler I: User information. Tech Summary Rep. 1469, Mathematics Res. Cent., Univ. Wisconsin–Madison, Madison, Wis., Dec. 1974 (revised April 1976)

Received December 1978, revised July 1979; accepted July 1979