

AD-A063 966

WISCONSIN UNIV-MADISON MATHEMATICS RESEARCH CENTER
AN AUGMENT INTERFACE FOR BRENT'S MULTIPLE PRECISION ARITHMETIC --ETC(U)
AUG 78 R P BRENT, J A HOOPER, J M YOHE
MRC-TSR-1868

F/G 9/2

DAAG29-75-C-0024

NL

UNCLASSIFIED

1 OF 1
ADA
063966

U.S. GOVT

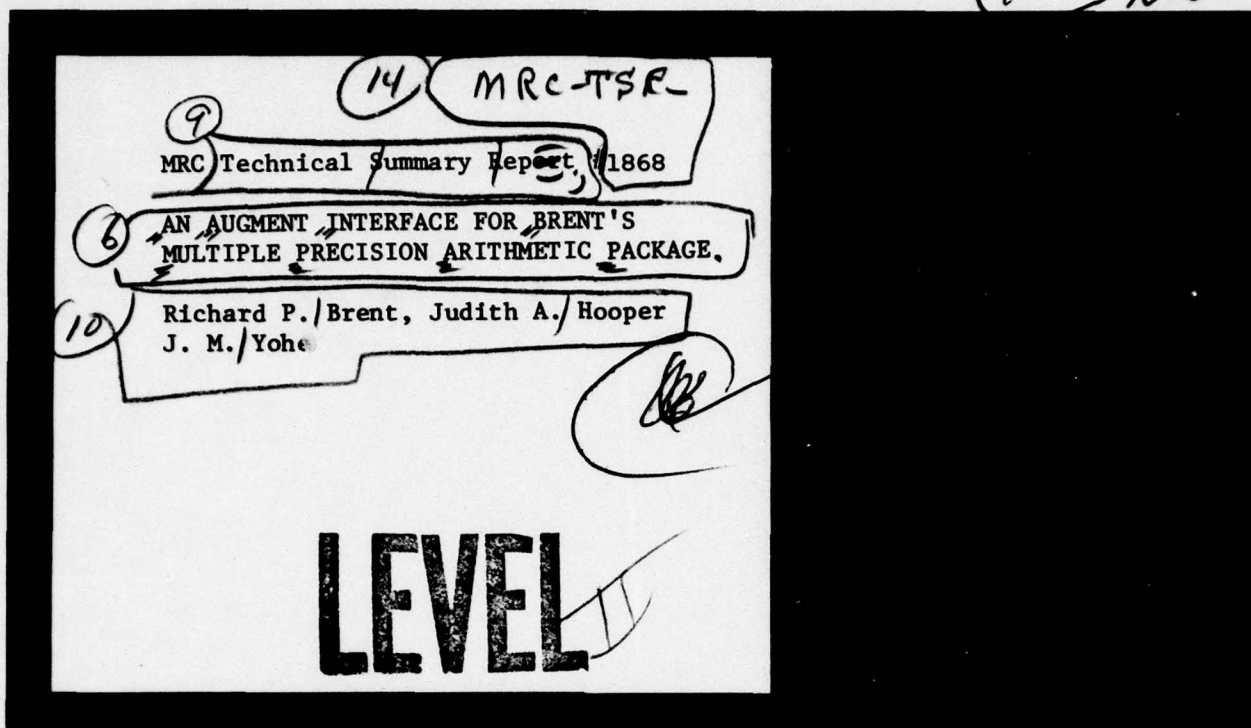


END
DATE
FILMED

3 -79
DDC

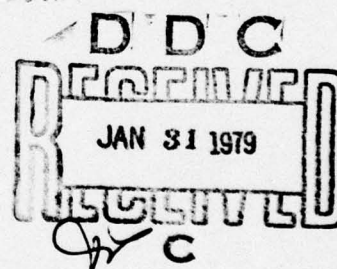
AD A063966

10 NW



15 DAAG29-75-C-0024

Mathematics Research Center
University of Wisconsin-Madison
610 Walnut Street
Madison, Wisconsin 53706



DDC FILE COPY

11 Aug 1978

12 26 p.

Received July 19, 1978

Approved for public release
Distribution unlimited

Sponsored by
U. S. Army Research Office
P.O. Box 12211
Research Triangle Park
North Carolina 27709

221200

79 01 30 063

mt

UNIVERSITY OF WISCONSIN - MADISON
MATHEMATICS RESEARCH CENTER

AN AUGMENT INTERFACE FOR BRENT'S
MULTIPLE PRECISION ARITHMETIC PACKAGE

Richard P. Brent (1), Judith A. Hooper (2), and J. M. Yohe (2)

Technical Summary Report #1868
August 1978

ABSTRACT

We describe the procedure required to interface the FORTRAN multiple precision package of Richard P. Brent (as described in ACM Transactions on Mathematical Software, March, 1978) with the AUGMENT precompiler for FORTRAN. We also indicate the method of using the multiple precision arithmetic package in conjunction with AUGMENT.

AMS(MOS) subject classification: 94-04

Computing Reviews Categories: 4.49, 5.11, 5.12

Key words: Arithmetic
Multiple precision
Extended precision
Floating point
Portable software
Software package
Precompiler interface
AUGMENT interface

Work Unit Number 8 - Computer Science

ACCESSION for	
NTIS	White Section ☒
DDC	Buff Section ☐
UNANNOUNCED	☐
JUSTIFICATION	
BY	
DISTRIBUTION/AVAILABILITY CODES	
Dist	SPECIAL
A	

- (1) Computing Research Group
Australian National University
Canberra, Australia
- (2) Mathematics Research Center
University of Wisconsin - Madison
Madison, Wisconsin 53706

SIGNIFICANCE AND EXPLANATION

In some applications, it is necessary to use higher precision than is afforded by standard software. The multiple precision arithmetic package developed by Richard P. Brent and described in the March, 1978 issue of ACM Transactions on Mathematical Software is extremely useful in such cases.

The disadvantages of using Brent's package directly are (1) the difficulty of converting existing programs to make use of the multiple precision package, and (2) the fact that in order to write a program using the package, one must parse the arithmetic expressions oneself and write the program as a series of calls on the package subroutines.

The AUGMENT precompiler for FORTRAN, developed at the Mathematics Research Center by F. D. Crary, is designed to simplify the use of packages such as Brent's. In this report, we describe the necessary interface to enable one to use Brent's package with AUGMENT, and provide instructions for its use.

AN AUGMENT INTERFACE FOR BRENT'S MULTIPLE PRECISION ARITHMETIC PACKAGE

Richard P. Brent, Judith A. Hooper, and J. M. Yohe

1. Introduction:

The purpose of this note is twofold: first, we demonstrate the ease with which a well-designed nonstandard arithmetic package may be interfaced with the AUGMENT precompiler for FORTRAN [4]; second, we provide an interface and user instructions to enable the reader to use Richard P. Brent's FORTRAN multiple precision arithmetic package [1], [2] in conjunction with AUGMENT. This makes the use of Brent's package 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. In only a few instances must the user write explicit calls on package modules; these cases will be discussed later in the paper.

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 gained by reading [4]. The supporting package to be interfaced with AUGMENT is the FORTRAN multiple precision arithmetic package described by Brent in [1] and [2]. This is a collection of portable subroutines which performs not only basic arithmetic operations, but also all of the ANSI standard mathematical functions and many nonstandard ones, in multiple precision. The precision of the package is governed entirely by the user at run time, and may even be changed during the course of a computation, provided the dimensions of the arrays reserved for the multiple precision numbers are not exceeded.

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; we address this question later.

The first step in interfacing the package was to prepare the AUGMENT description decks. The multiple precision arithmetic package (although not designed with an AUGMENT interface in mind) was extremely compatible with AUGMENT: most of the multiple precision routines were cast as 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.

The preparation of the description deck therefore proceeded easily; 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 [5]. 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 we shall discuss 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 resulting description deck is shown in Appendix B.

The routines to generate constants posed a small problem: AUGMENT assumes that routines will have at least one argument in addition to the result (this might be regarded as a deficiency in AUGMENT), and these routines did not. We therefore decided to write a short routine to interface these routines with AUGMENT, casting it as a conversion routine which "converts" the Hollerith name of the desired constant to the value of the constant. This routine is called with the Hollerith name of the constant as an argument (e.g., 'PI'), unpacks this Hollerith string, determines which of the constant-generating routines to call, and returns the resulting value. Once this routine was written, it seemed logical to include the capability of run-time conversion of numeric constants, so we extended the routine by adding a call to another package routine to convert the (presumably numeric) Hollerith string to multiple precision if it did not match the name of any of the "standard" 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. (These should be modified only with extreme care, however.) 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 of them is given in Appendix C.

In order to interface the PACK and UNPK routines, we introduced another data type called MULTIPAK; the PACK and UNPK routines were then described as conversions between types MULTIPLE and MULTIPAK.

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

3. Use of the package via AUGMENT:

As explained in [4], 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 through 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.

If it is desired to store multiple precision variables in packed form, one would declare a 10 by 100 array of packed variables in the following way:

```
MULTIPAK A(10,100)
```

Since the package normally operates only on unpacked variables, any packed variables must normally be converted to unpacked format before use. This may be accomplished by either of two methods:

```
X = A(I, J)      (normal replacement statement)
CTM(A(I, J))     (conversion function)
```

Packed variables should not normally be used directly in arithmetic expressions, since AUGMENT will not generate the appropriate conversion in all cases. Packed variables may be used in certain expressions; for example, if A and C are type MULTIPLE and B is type MULTIPAK, the expression

```
A = B * C
```

will work properly. However, the expression

```
A = EXP(B)
```

will not work; it must be written as

```
A = EXP(CTM(B)).
```

The user may elect to try mixed mode expressions of other kinds; the worst that can happen is that the linkage editor will discover that AUGMENT has generated a call on a nonexistent routine.

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. If the compiler generates a sentinel, and if the unpacking routine recognizes it, the terminal '\$' is unnecessary. (Note that the Hollerith-unpacking routine is NOT portable; it will need to be rewritten for each new system. The ones shown in Appendix C are for UNIVAC 1100 FORTRAN V, UNIVAC 1100 ASCII FORTRAN, and IBM 360 FORTRAN G or H, respectively.)

The user must still 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 method of setting the parameters is by including the following statements in the main program, before any (other) executable statements:

```
COMMON IDUMMY(K) (where K = MAXR + 5)
CALL MPSET (LUN, NDIGIT, N, MAXR)
```

where LUN is the logical unit number for output (usually 6); NDIGIT is the number of decimal digits of precision desired; N is the number of storage locations required for each multiple precision variable (this must not exceed the number given in Line 23 of the description deck -- 12 in the deck shown in Appendix B); and MAXR is the length of the working space array as described in [1]. Of course, the user may also set these parameters directly as described in [3], but in that case, care must be exercised not to exceed the number of locations assigned to variables by AUGMENT.

Another way of setting these parameters to default values is to include the statement

```
INITIALIZE MP
```

in the type declarations. This causes AUGMENT to generate a call on the routine MPINIT, which then sets the parameters to values fixed in the MPINIT subroutine. Of course, changes in the dimensions in the description deck must be accompanied by appropriate changes in the parameters in MPINIT if this method is to be used. This is a bit of a cludge, but it works, provided the default values are what one really wants.

A third way of providing these parameters to the package would be even easier, but would require some modification of the package. If all occurrences of blank COMMON were changed to labeled COMMON (e.g., COMMON/MPCOM/), the package parameters could then be set via a DATA statement. (This was not done in the existing package because of a restriction in the ANSI (1966) standards; according to these standards, labeled COMMON must be declared consistently in all routines.) The setting of these parameters in this manner would obviate the need for the user to take any action at all; however, it would result in incompatibility with the standard (published) version of the multiple precision arithmetic package.

The maximum precision available to the user via the given description deck depends on the word length of the host computer; on the UNIVAC 1110, it is approximately 43 digits. The value of MAXR likewise depends on the characteristics of the host machine (and on the modules of the package being used in the program); we used 296 for the UNIVAC 1110 assuming 12 words per multiple precision variable.

If the precision provided by the description deck is not sufficient for the user's needs, it is not a difficult task to increase it; one merely increases the value of N given in Line 23 of the description deck to accommodate the desired precision; increases the value in Line 20 of the description deck to $\text{INT}((N + 1)/2)$; and increases MAXR as appropriate. The number of locations needed for the work space array will depend on which of the package routines are being used; the amount of work space needed for each routine is given in [3]. The most space-consuming routines are MPBESJ and MPLNGM. If one wishes to avoid the pain of calculating the precise requirements, one may be assured that by using

$$\text{MAXR} = \max(T^2 + 15T + 27, 14T + 156)$$

where $T = N - 2$, enough work space will be reserved for any routine in the package. These considerations are discussed in greater detail in [3].

Once the program has been written, the following runstream will invoke AUGMENT and cause the translated program to be written on logical unit 20:

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

The resulting program on logical unit 20 would then be compiled just like any other FORTRAN program; the compiled program would then be 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 shown in Appendix A.

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.

The interface shown in this paper is self-contained, and can be used (with appropriate modifications, as indicated in the text) with Brent's multiple precision arithmetic package, assuming AUGMENT is available. A revised version of the multiple precision arithmetic package, incorporating the AUGMENT interface routines, is available from the first author.

Questions may be addressed to the authors.

REFERENCES

1. Brent, Richard P., A FORTRAN multiple-precision arithmetic package, Assoc. Comput. Mach. Trans. Math. Software 4 (1978), 57-70.
2. Brent, Richard P., Algorithm 524, MP, a FORTRAN multiple-precision arithmetic package, Assoc. Comput. Mach. Trans. Math. Software 4 (1978), 71-81.
3. Brent, Richard P., MP users guide, Australian National University, Canberra, Australia, Computer Centre, Technical Report #54, September, 1976 (revised July, 1978).
4. Crary, F. D., A versatile precompiler for nonstandard arithmetics, Assoc. Comput. Mach. Trans. Math. Software (to appear).
5. Crary, F. D., The AUGMENT precompiler I: User information, The University of Wisconsin - Madison, Mathematics Research Center, Technical Summary Report #1469, December, 1974 (revised April, 1976).

APPENDIX A OPERATIONS IMPLEMENTED IN BRENT'S MULTIPLE PRECISION PACKAGE

OPERATION	DEFINITION/EXPLANATION	RESULT TYPE	VIA AUGMENT	ROUTINE INVOCATION	ROUTINE TYPE (AUGMENT)
ARITHMETIC					
ADDITION					
	Sum of two MP numbers	M	MA + MB	MPADD(MA, MB, MR)	S
	Sum of MP number and an integer	M	MA + IB	MPADDI(MA, IB, MR)	S
	Sum of MP number and the rational number IB/IC	M	ADDQ(MA, IB, IC)	MPADDQ(MA, IB, IC, MR)	S
DIVISION					
	Quotient of two MP numbers	M	MA / MB	MPDIV(MA, MB, MR)	S
	Quotient of MP number and an integer	M	MA / IB	MPDIVI(MA, IB, MR)	S
MULTIPLICATION					
	Product of two MP numbers	M	MA * MB	MPMUL(MA, MB, MR)	S
	Product of MP number and an integer	M	MA * IB	MPMULI(MA, IB, MR)	S
	Product of MP number and rat. number IB/IC	M	MULQ(MA, IB, IC)	MPMULQ(MA, IB, IC, MR)	S
RECIPROCAL					
	Reciprocal of MP number	M	REC(MA)	MPREC(MA, MR)	S
SUBTRACTION					
	Difference of two MP numbers	M	MA - MB	MPSUB(MA, MB, MR)	S
POWERS AND ROOTS					
	Raise MP number to integer power	M	MA ** IB	MPPWR(MA, IB, MR)	S
	Raise MP number to MP power	M	MA ** MB	MPMPWR2(MA, MB, MR)	S
	Raise rat. number IA/IB to rat. power IC/ID	M	QWR(IA, IB, IC, ID)	MPQWR(IA, IB, IC, ID, MR)	S
	IBth root of MP number	M	ROOT(MA, IB)	MPROOT(MA, IB, MR)	S
ELEMENTARY FUNCTIONS					
	Absolute value	M	ABS(M)	MPABS(MA, MR)	S
	Arc sine of MP number	M	ASIN(MA)	MPASIN(MA, MR)	S
	Arc tangent of MP number	M	ATAN(MA)	MPATAN(MA, MR)	S
	Arc tangent of 1/IA	M	ARTI(IA)	MPARTI(IA, MR)	S
	Cosine of MP number	M	COS(MA)	MPCOS(MA, MR)	S
	Hyperbolic cosine of MP number	M	COSH(MA)	MPCOSH(MA, MR)	S
	Exp of MP number	M	EXP(MA)	MPEXP(MA, MR)	S
	(Exp - 1) of MP number	M	EXP1(MA)	MPEXP1(MA, MR)	S
	Fractional part of MP number	M	FRAC(MA)	MPCFRAC(MA, MR)	S
	Integer part of MP number	M	INT(MA)	MPCMIN(MA, MR)	S
	Natural logarithm of MP number	M	LOG(MA)	MPLN(MA, MR)	S
	Natural logarithm of MP number using Gauss-Salamini algorithm	M	LNGS(MA)	MPLNGS(MA, MR)	S
	Natural logarithm of (1 + MP number)	M	LNS(MA)	MPLNS(MA)	S
	Natural logarithm of small positive integer	M	LOG(IA)	MPLNI(IA, MR)	S
	Maximum of two MP numbers	M	MAX(MA, MB)	MPMAX(MA, MB, MR)	S
	Minimum of two MP numbers	M	MIN(MA, MB)	MPMIN(MA, MB, MR)	S
	Sine of MP number	M	SIN(MA)	MPSIN(MA, MR)	S
	Hyperbolic sine of MP number	M	SINH(MA)	MPSINH(MA, MR)	S
	Square root of MP number	M	SQRT(MA)	MPSQRT(MA, MR)	S
	Tangent of MP number	M	TAN(MA)	MPTAN(MA, MR)	S
	Hyperbolic tangent of MP number	M	TANH(MA)	MPTANH(MA, MR)	S

APPENDIX A (Continued) OPERATIONS IMPLEMENTED IN BRENT'S MULTIPLE PRECISION PACKAGE

OPERATION	DEFINITION/EXPLANATION	RESULT TYPE	VIA AUGMENT	ROUTINE INVOCATION DIRECT	ROUTINE TYPE (AUGMENT)
SPECIAL FUNCTIONS					
	Bessel function (first kind) of integer order	M	BESJ(MA, IB)	MPBESJ(MA, IB, MR)	S
	Dawson's integral of MP argument	M	DAW(MA)	MPDAW(MA, MR)	S
	Exponential integral of MP argument	M	EI(MA)	MPEI(MA, MR)	S
	Error function of MP number	M	ERF(MA)	MPERF(MA, MR)	S
	Complementary error function of MP number	M	ERFC(MA)	MPERFC(MA, MR)	S
	Gamma function of MP number	M	GAM(MA)	MPGAM(MA, MR)	S
	Gamma function of rational number IA/IB	M	GAM(IA, IB)	MPGAMQ(IA, IB, MR)	S
	GCD of two MP integers	M	GCD(MA, MB)	MPGCD(MA, MB, MR)	S
	Logarithmic integral of MP number	M	LI(MA)	MPLI(MA, MR)	S
	Logarithm of gamma function of MP number	M	LNGM(MA)	MPLNGM(MA, MR)	S
	Riemann zeta function of pos. integer	M	ZETA(IA)	MPZETA(IA, MR)	S
CONSTANTS					
	Bernoulli numbers	PM	BERN(IA, IB)	MPBERN(IA, IB, PMR)	S
	Multiple precision machine precision	M	CTM('EPS')	MPEPS(MR)	S
	Euler's constant	M	CTM('EUL')	MPEUL(MR)	S
	Largest positive MP number	M	CTM('MAXR')	MPMAXR(MR)	S
	Smallest positive MP number	M	CTM('MINR')	MPMINR(MR)	S
	Pi	M	CTM('PI')	MPPI(MR)	S
INPUT/OUTPUT					
	Read IB words from Unit IC, under Format HD, convert to MP number MR; LR is error code.	L	MPINF(MR, IB, IC, HD)	MPINF(MR, IB, IC, HD, LR)	S
	Write IB words, representing MP number MA, on Unit LUN, IC places after decimal point, under format HD; LR is error code.	L	MPOUTF(MA, IB, IC, HD)	MPOUTF(MA, IB, IC, HD, LR)	S
	Dump MP number on Logical Unit LUN	-	-	MPDUMP(MA)	S
	Convert unpacked Hollerith fixed point to MP	M	-	MPIN(UHA, MR, IB, IR)	S
	Convert upkd Hol fixed pt. + exp IC to MP	M	-	MPINE(UHA, MR, IB, IC, IR)	S
	Convert MP to upkd Hol (fixed pt.)	UH	-	MPOUT(MA, UHR, IC, ID)	S
	Convert MP to upkd Hol. (floating pt.)	UH, I	-	MPOUTE(MA, UHR, IR, ID)	S
CONVERSION					
	Double Precision to Multiple	M	CTM(DA)	MPCDM(DA, MR)	S
	Integer to Multiple	M	CTM(IA)	MPCIM(IA, MR)	S
	Real to Multiple	M	CTM(RA)	MPCRM(RA, MR)	S
	Rational IA/IB to Multiple	M	CTM(IA, IB)	MPCQM(IA, IB, MR)	S
	Packed Multiple to Multiple	M	CTM(PMA)	MPUNPK(PMA, MR)	S
	Packed Hollerith to Multiple	M	CTM(HA)	MPCAM(HA, MR)	S
	Multiple to Double Precision	D	CTD(MA)	MPCMD(MA, DR)	S
	Multiple to Integer	I	CTI(MA)	MPCMI(MA, IR)	S
	Multiple to Real	R	CTR(MA)	MPCMR(MA, IR)	S
	Multiple to Packed Multiple	PM	CTP(MA)	MPPACK(MA, PR)	S
	Multiple to Double Precision + Integer exponent	D, I	-	MPCMDE(MA, IR, DR)	S
	Multiple to Multiple + Integer exponent	M, I	-	MPCMEF(MA, IR, MR)	S
	Multiple to Real + Integer exponent	M, I	-	MPCMRE(MA, IR, RR)	S

APPENDIX A (Continued)

OPERATIONS IMPLEMENTED IN BRENT'S MULTIPLE PRECISION PACKAGE

OPERATION	DEFINITION/EXPLANATION	RESULT TYPE	ROUTINE INVOCATION	ROUTINE TYPE (AUGMENT)
COMPARISON	(+1 if >, 0 if =, -1 if <)			
	Compare absolute value of MP numbers			
	Compare MP number with integer			
	Compare MP number with real			
RELATIONAL	Compare MP numbers			
	MA equal to MB			
	MA greater than or equal to MB			
	MA greater than MB			
TEST	MA less than or equal to MB			
	MA less than MB			
	MA not equal to MB			
	Three-way branch			
FIELD FUNCTIONS	Sign of MP number			
	Exponent of MP number			
	10th digit of MP number			
	Number of MP digits			
UTILITY	Maximum exponent of MP number			
	MP base			
	Unary minus			
	Replacement			
ERROR DETECTION	Evaluate Polynomial (integer coefs, dim IC)			
	Set parameters for MP routines			
	Clear next IB positions of MP number			
	Check legality of parameters to MP package			
	Handle fatal error conditions			
	Handle MP overflow			
	Handle MP underflow			

APPENDIX A (Continued)
OPERATIONS IMPLEMENTED IN BRENT'S MULTIPLE PRECISION PACKAGE

NOTES ON TABLE:

1. Data types are indicated by one- or two-letter abbreviations: D = DOUBLE PRECISION; H = PACKED HOLLERITH; I = INTEGER; L = LOGICAL; M = MULTIPLE; PM = PACKED MULTIPLE; R = REAL; UH = UNPACKED HOLLERITH.
2. Variable names: The first letter (or pair of letters) indicates the data type of the variable as above. The terminal letter is A, B, C, or D for an argument; R for result.
3. Routine types: S denotes subroutine; any other letter denotes a function of the designated type.
4. A field function is one which allows specified portions of a data element to be altered or retrieved selectively. Extreme care should be used in altering fields of data elements.
5. The conversion routines (those beginning with 'CT' in Column 4) may also be invoked implicitly via replacement statements. For example, the statement "MR = DA" will cause AUGMENT to generate a call on MPCDM as shown in Column 5.
6. In some cases, AUGMENT will recognize synonyms of the names given in Column 4. Particulars may be found in the description deck (Appendix B). Of course, the user may change or add to the recognition names by modifying the description deck; see [5] for details.

APPENDIX B

AUGMENT DESCRIPTION DECK FOR BRENT'S MULTIPLE PRECISION ARITHMETIC PACKAGE

*DESCRIBE MULTIPAK	MPA00010
COMMENT AUGMENT DESCRIPTION DECK FOR THE MULTIPLE-PRECISION	MPA00020
ARITHMETIC PACKAGE OF R. P. BRENT, UNIVAC 1100 VERSION.	MPA00030
THREE TYPES OF VARIABLE ARE DEFINED HERE -	MPA00040
MULTIPLE (STANDARD MULTIPLE-PRECISION NUMBERS),	MPA00050
MULTIPAK (PACKED MULTIPLE-PRECISION NUMBERS), AND	MPA00060
INITIALIZE (USED ONLY AS A DEVICE TO PERSUADE	MPA00070
AUGMENT TO INITIALIZE THE MP PACKAGE).	MPA00080
WORKING SPACE SHOULD BE ALLOCATED AND THE MP PACKAGE	MPA00090
INITIALIZED BY THE DECLARATION	MPA00100
INITIALIZE MP	MPA00110
IN THE MAIN PROGRAM.	MPA00120
THIS DESCRIPTION DECK ASSUMES THAT MULTIPLE PRECISION NUMBERS	MPA00130
WILL HAVE NO MORE THAN 10 DIGITS (BASE 65536) FOR A TOTAL	MPA00140
PRECISION NOT EXCEEDING ABOUT 43 DECIMAL PLACES. FOR THIS,	MPA00150
EACH MP NUMBER REQUIRES 12 WORDS (6 IN PACKED FORMAT).	MPA00160
SEE COMMENTS IN ROUTINE MPINIT FOR THE METHOD OF CHANGING	MPA00170
THE PRECISION OR ADAPTING TO A MACHINE WITH WORDLENGTH	MPA00180
OTHER THAN 36 BITS.	MPA00190
DECLARE INTEGER(6), KIND SAFE SUBROUTINE, PREFIX MPK	MPA00200
SERVICE COPY(STR)	MPA00210
*DESCRIBE MULTIPLE	MPA00220
DECLARE INTEGER(12), KIND SAFE SUBROUTINE, PREFIX MP	MPA00230
OPERATOR + (, NULL UNARY, PRV, \$), - (NEG, UNARY),	MPA00240
+ (ADD, BINARY3, PRV, \$, \$, \$, COMM), * (MUL),	MPA00250
- (SUB, , , , , NONCOMM), / (DIV), ** (PWR2),	MPA00260
+ (ADDI, , , , INTEGER), * (MULI), / (DIVI), ** (PWR),	MPA00270
.EQ. (EQ, BINARY2, PRV, \$, LOGICAL, COMM), .NE. (NE),	MPA00280
.GE. (GE, , , , , NONCOMM), .GT. (GT), .LE. (LE), .LT. (LT)	MPA00290
TEST MPSIGA (SIGA, INTEGER)	MPA00300
FIELD SGN (SIGA, SIGB, (\$), INTEGER),	MPA00310
EXPON (EXPA, EXPB), BASE (BASA, BASB), NUMDIG (DIGA, DIGB),	MPA00320
MAXEXP (MEXA, MEXB), DIGIT (DGA, DGB, (\$), INTEGER)	MPA00330
FUNCTION ABS (ABS, (\$), \$), ASIN (ASIN), ATAN (ATAN), CMF (CMF),	MPA00340
CMIM (CMIM), COS (COS), COSH (COSH), DAW (DAW), EI (EI),	MPA00350
ERF (ERF), ERFC (ERFC), EXP (EXP), EXP1 (EXP1), FRAC (CMF),	MPA00360
GAM (GAM), INT (CMIM), LI (LI), LN (LN), LOG (LN), LNGM (LNGM),	MPA00370
LNGS (LNGS), LNS (LNS), REC (REC), SIN (SIN), SINH (SINH),	MPA00380
SQRT (SQRT), TAN (TAN), TANH (TANH),	MPA00390
ART1 (ART1, (INTEGER)), LN (LNI), LNI (LNI), LOG (LNI),	MPA00400
ZETA (ZETA), CAM (CAM), CAM (CAM, (HOLLERITH)),	MPA00410
MAX (MAX, (\$, \$)), MIN (MIN), GCD (GCDA),	MPA00420
BESJ (BESJ, (\$, INTEGER)), ROOT (ROOT),	MPA00430
MPINF (INF(SUBROUTINE), (\$, INTEGER, INTEGER, HOLLERITH), LOGICAL),	MPA00440
MPOUTF (OUTF(SUBROUTINE)),	MPA00450
MPINF (INF(SUBROUTINE), (\$, INTEGER, INTEGER, INTEGER)),	MPA00460
MPOUTF (OUTF(SUBROUTINE)),	MPA00470
COMP (COMP, (\$, \$), INTEGER), CMPA (CMPA),	MPA00480
COMP (CMPI, (\$, INTEGER)), COMP (CMPR, (\$, REAL)),	MPA00490
ADDQ (ADDQ, (\$, INTEGER, INTEGER), \$), MULQ (MULQ),	MPA00500

QPWR (QPWR, (INTEGER, INTEGER, INTEGER, INTEGER)),	MPA00510
CQM (CQM, (INTEGER, INTEGER)), CTM (CQM),	MPA00520
GAM (GAMQ), GAMQ (GAMQ),	MPA00530
BERN (BERN, (INTEGER, INTEGER), MULTIPAK)	MPA00540
CONVERSION CTM (CDM, DOUBLE PRECISION, \$, UPWARD),	MPA00550
CTM (CIM, INTEGER), CTM (CRM, REAL),	MPA00560
CTM (UNPK, MULTIPAK), CTM (CAM, HOLLERITH),	MPA00570
CTD (CMD(SUBROUTINE), \$, DOUBLE PRECISION, DOWNWARD),	MPA00580
CTI (CMI(SUBROUTINE),, INTEGER),	MPA00590
CTR (CMR(SUBROUTINE),, REAL), CTP (PACK,, MULTIPAK)	MPA00600
SERVICE COPY (STR)	MPA00610
*DESCRIBE INITIALIZE	MPA00620
DECLARE INTEGER(1), KIND SAFE SUBROUTINE, PREFIX MPI	MPA00630
SERVICE COPY (STR), INITIAL (NIT)	MPA00640
COMMENT END OF AUGMENT DESCRIPTION DECK FOR MP PACKAGE	MPA00650
	MPA00660

APPENDIX C

AUGMENT INTERFACE ROUTINES FOR BRENT'S MULTIPLE PRECISION ARITHMETIC PACKAGE

C \$\$	***** MPBASA *****	MP009551
	FUNCTION MPBASA (X)	MP009553
C	RETURNS THE MP BASE (FIRST WORD IN COMMON).	MP009555
C	X IS A DUMMY MP ARGUMENT.	MP009557
	COMMON B, T, M, LUN, MXR, R	MP009559
	INTEGER B, T, R(1), X(1)	MP009561
	MPBASA = B	MP009563
	RETURN	MP009565
	END	MP009567
C \$\$	***** MPBASB *****	MP009571
	SUBROUTINE MPBASB (I, X)	MP009573
C	SETS THE MP BASE (FIRST WORD OF COMMON) TO I.	MP009575
C	I SHOULD BE AN INTEGER SUCH THAT I .GE. 2	MP009577
C	AND (8*I-1) IS REPRESENTABLE AS A SINGLE-PRECISION INTEGER.	MP009579
C	X IS A DUMMY MP ARGUMENT (AUGMENT EXPECTS ONE).	MP009581
	COMMON B, T, M, LUN, MXR, R	MP009583
	INTEGER B, T, R(1), X(1)	MP009585
C	SET BASE TO I, THEN CHECK VALIDITY	MP009587
	B = I	MP009589
	CALL MPCHK (1, 4)	MP009591
	RETURN	MP009593
	END	MP009595
C \$\$	***** MPCAM *****	MP012491
	SUBROUTINE MPCAM (A, X)	MP012493
C	CONVERTS THE HOLLERITH STRING A TO AN MP NUMBER X.	MP012495
C	A CAN BE A STRING OF DIGITS ACCEPTABLE TO ROUTINE MPIN	MP012497
C	AND TERMINATED BY A DOLLAR (\$), E.G. 7H-5.367\$,	MP012499
C	OR ONE OF THE FOLLOWING SPECIAL STRINGS -	MP012501
C	EPS (MP MACHINE-PRECISION, SEE MPEPS),	MP012503
C	EUL (EULERS CONSTANT 0.5772..., SEE MPEUL),	MP012505
C	MAXR (LARGEST VALID MP NUMBER, SEE MPMAXR),	MP012507
C	MINR (SMALLEST POSTIVE MP NUMBER, SEE MPMINR),	MP012509
C	PI (PI = 3.14..., SEE MPPI).	MP012511
C	ONLY THE FIRST TWO CHARACTERS OF THESE STRINGS ARE CHECKED.	MP012513
C	SPACE REQUIRED IS NO MORE THAN 5*T+L+14, WHERE L IS THE	MP012515
C	NUMBER OF CHARACTERS IN THE STRING A (EXCLUDING \$).	MP012517
C	IF SPACE IS LESS 3*T+L+11 THE STRING A WILL EFFECTIVELY BE TRUNCATED	MP012519
	COMMON B, T, M, LUN, MXR, R	MP012521
	INTEGER B, T, R(1), A(1), X(1), ERROR, C(6), D(2)	MP012523
	DATA C(1) /1HA/, C(2) /1HE/, C(3) /1HI/	MP012525
	DATA C(4) /1HM/, C(5) /1HP/, C(6) /1HU/	MP012527
C	UNPACK FIRST 2 CHARACTERS OF A	MP012529
	CALL MPUPK (A, D, 2, N)	MP012531
	IF (N.NE.2) GO TO 10	MP012533
C	SET X TO ZERO AFTER SAVING A(1) IN CASE A AND X COINCIDE	MP012535
	I = A(1)	MP012537
	X(1) = 0	MP012539
C	CHECK FOR SPECIAL STRINGS	MP012541

IF ((D(1).EQ.C(2)).AND.(D(2).EQ.C(5))) CALL MPEPS (X)	MP012543
IF ((D(1).EQ.C(2)).AND.(D(2).EQ.C(6))) CALL MPEUL (X)	MP012545
IF ((D(1).EQ.C(4)).AND.(D(2).EQ.C(1))) CALL MPMAXR (X)	MP012547
IF ((D(1).EQ.C(4)).AND.(D(2).EQ.C(3))) CALL MPMINR (X)	MP012549
IF ((D(1).EQ.C(5)).AND.(D(2).EQ.C(3))) CALL MPPI (X)	MP012551
C RETURN IF X NONZERO (SO ONE OF ABOVE TESTS SUCCEEDED)	MP012553
IF (X(1).NE.0) RETURN	MP012555
C RESTORE A(1) AND UNPACK, THEN CALL MPIN TO DECODE.	MP012557
A(1) = I	MP012559
10 I2 = 3*T + 12	MP012561
CALL MPUPK (A, R(I2), MXR+1-I2, N)	MP012563
CALL MPIN (R(I2), X, N, ERROR)	MP012565
IF (ERROR.EQ.0) RETURN	MP012567
WRITE (LUN, 20)	MP012569
20 FORMAT (53H *** ERROR IN HOLLERITH CONSTANT IN CALL TO MPCAM ***)	MP012571
CALL MPERR	MP012573
RETURN	MP012575
END	MP012577
C \$\$ ***** MPDGA *****	MP019741
FUNCTION MPDGA (X, N)	MP019743
C RETURNS THE N-TH DIGIT OF THE MP NUMBER X FOR 1 .LE. N .LE. T.	MP019745
C RETURNS ZERO IF X IS ZERO OR N .LE. 0 OR N .GT. T.	MP019747
COMMON B, T, M, LUN, MXR, R	MP019749
INTEGER B, T, R(1), X(1)	MP019751
MPDGA = 0	MP019753
IF ((X(1).NE.0).AND.(N.GT.0).AND.(N.LE.T)) MPDGA = X(N+2)	MP019755
RETURN	MP019757
END	MP019759
C \$\$ ***** MPDGB *****	MP019781
SUBROUTINE MPDGB (I, X, N)	MP019783
C SETS THE N-TH DIGIT OF THE MP NUMBER X TO I.	MP019785
C N MUST BE IN THE RANGE 1 .LE. N .LE. T,	MP019787
C I MUST BE IN THE RANGE 0 .LE. I .LT. B	MP019789
C (AND I .NE. 0 IF N .EQ. 1).	MP019791
C THE SIGN AND EXPONENT OF X ARE UNCHANGED.	MP019793
COMMON B, T, M, LUN, MXR, R	MP019795
INTEGER B, T, R(1), X(1)	MP019797
IF ((N.GT.0).AND.(N.LE.T)) GO TO 20	MP019799
WRITE (LUN, 10)	MP019801
10 FORMAT (48H *** DIGIT POSITION ILLEGAL IN CALL TO MPDGB ***)	MP019803
GO TO 40	MP019805
20 IF ((I.GE.0).AND.(I.LT.B).AND.((I+N).GT.1)) GO TO 50	MP019807
WRITE (LUN, 30)	MP019809
30 FORMAT (45H *** DIGIT VALUE ILLEGAL IN CALL TO MPDGB ***)	MP019811
40 CALL MPERR	MP019813
RETURN	MP019815
50 X(N+2) = I	MP019817
RETURN	MP019819
END	MP019821
C \$\$ ***** MPDIGA *****	MP019841
FUNCTION MPDIGA (X)	MP019843
C RETURNS THE NUMBER OF MP DIGITS (SECOND WORD IN COMMON).	MP019845
C X IS A DUMMY MP ARGUMENT.	MP019847

COMMON B, T, M, LUN, MXR, R	MP019849
INTEGER B, T, R(1), X(1)	MP019851
MPDIGA = T	MP019853
RETURN	MP019855
END	MP019857
C \$\$ ***** MPDIGB *****	MP019861
SUBROUTINE MPDIGB (I, X)	MP019863
C SETS THE NUMBER OF MP DIGITS (SECOND WORD OF COMMON) TO I.	MP019865
C I SHOULD BE AN INTEGER SUCH THAT I .GE. 2	MP019867
C X IS A DUMMY MP ARGUMENT (AUGMENT EXPECTS ONE).	MP019869
C WARNING *** MP NUMBERS MUST BE DECLARED AS INTEGER ARRAYS OF	MP019871
C *** DIMENSION AT LEAST I+2. MPDIGB DOES NOT CHECK THIS.	MP019873
COMMON B, T, M, LUN, MXR, R	MP019875
INTEGER B, T, R(1), X(1)	MP019877
C SET DIGITS TO I, THEN CHECK VALIDITY	MP019879
T = I	MP019881
CALL MPCHK (1, 4)	MP019883
RETURN	MP019885
END	MP019887
C \$\$ ***** MPEQ *****	MP023221
LOGICAL FUNCTION MPEQ (X, Y)	MP023223
C RETURNS LOGICAL VALUE OF (X .EQ. Y) FOR MP X AND Y.	MP023225
INTEGER X(1), Y(1)	MP023227
MPEQ = (MPCOMP(X,Y) .EQ. 0)	MP023229
RETURN	MP023231
END	MP023233
C \$\$ ***** MPEXPA *****	MP027271
FUNCTION MPEXPA (X)	MP027273
C RETURNS THE EXPONENT OF THE MP NUMBER X	MP027275
C (OR LARGEST NEGATIVE EXPONENT IF X IS ZERO).	MP027277
COMMON B, T, M, LUN, MXR, R	MP027279
INTEGER B, T, R(1), X(2)	MP027281
MPEXPA = -M	MP027283
C RETURN -M IF X ZERO, X(2) OTHERWISE	MP027285
IF (X(1).NE.0) MPEXPA = X(2)	MP027287
RETURN	MP027289
END	MP027291
C \$\$ ***** MPEXPB *****	MP027311
SUBROUTINE MPEXPB (I, X)	MP027313
C SETS EXPONENT OF MP NUMBER X TO I UNLESS X IS ZERO	MP027315
C (WHEN EXPONENT IS UNCHANGED).	MP027317
C X MUST BE A VALID MP NUMBER (EITHER ZERO OR NORMALIZED).	MP027319
COMMON B, T, M, LUN, MXR, R	MP027321
INTEGER B, T, R(1), X(3)	MP027323
C RETURN IF X IS ZERO	MP027325
IF (X(1).EQ.0) RETURN	MP027327
C CHECK FOR VALID MP SIGN AND LEADING DIGIT	MP027329
IF ((IABS(X(1)).LE.1).AND.(X(3).GT.0).AND.(X(3).LT.B))	MP027331
\$ GO TO 20	MP027333
WRITE (LUN, 10)	MP027335
10 FORMAT (48H *** X NOT VALID MP NUMBER IN CALL TO MPEXPB ***)	MP027337
CALL MPERR	MP027339

X(1) = 0	MP027341
RETURN	MP027343
C SET EXPONENT OF X TO I	MP027345
20 X(2) = I	MP027347
C CHECK FOR OVERFLOW AND UNDERFLOW	MP027349
IF (I.GT.M) CALL MPOVFL (X)	MP027351
IF (I.LT.(-M)) CALL MPUNFL (X)	MP027353
RETURN	MP027355
END	MP027357
 C \$\$	 MP030521
***** MPGE *****	
LOGICAL FUNCTION MPGE (X, Y)	MP030523
C RETURNS LOGICAL VALUE OF (X .GE. Y) FOR MP X AND Y.	MP030525
INTEGER X(1), Y(1)	MP030527
MPGE = (MPCOMP(X,Y) .GE. 0)	MP030529
RETURN	MP030531
END	MP030533
 C \$\$	 MP030541
***** MPGT *****	
LOGICAL FUNCTION MPGT (X, Y)	MP030543
C RETURNS LOGICAL VALUE OF (X .GT. Y) FOR MP X AND Y.	MP030545
INTEGER X(1), Y(1)	MP030547
MPGT = (MPCOMP(X,Y) .GT. 0)	MP030549
RETURN	MP030551
END	MP030553
 C \$\$	 MP032761
***** MPINF *****	
SUBROUTINE MPINF (X, N, UNIT, IFORM, ERR)	MP032763
C READS N WORDS FROM LOGICAL UNIT IABS(UNIT) USING FORMAT IN IFORM,	MP032765
C THEN CONVERTS TO MP NUMBER X USING ROUTINE MPIN.	MP032767
C IFORM SHOULD CONTAIN A FORMAT WHICH ALLOWS FOR READING N WORDS	MP032769
C IN A1 FORMAT, E.G. 6H(80A1)	MP032771
C ERR RETURNED AS TRUE IF MPIN COULD NOT INTERPRET INPUT AS	MP032773
C AN MP NUMBER OR IF N NOT POSITIVE, OTHERWISE FALSE.	MP032775
C IF ERR IS TRUE THEN X IS RETURNED AS ZERO.	MP032777
C SPACE REQUIRED 3T+N+11.	MP032779
COMMON B, T, M, LUN, MXR, R	MP032781
INTEGER B, T, R(1), X(1), UNIT, IFORM(1)	MP032783
LOGICAL ERR	MP032785
C CHECK THAT ENOUGH SPACE AVAILABLE	MP032787
CALL MPCHK (3, N+11)	MP032789
I2 = 3*T + 12	MP032791
C READ N WORDS UNDER FORMAT IFORM.	MP032793
CALL MPIO (R(I2), N, (-IABS(UNIT)), IFORM, ERR)	MP032795
X(1) = 0	MP032797
C RETURN IF ERROR	MP032799
IF (ERR) RETURN	MP032801
C ELSE CONVERT TO MP NUMBER.	MP032803
CALL MPIN (R(I2), X, N, IER)	MP032805
C RETURN ERROR FLAG IF MPIN OBJECTED	MP032807
ERR = (IER.NE.0)	MP032809
RETURN	MP032811
END	MP032813
 C \$\$	 MP032821
***** MPINIT *****	
SUBROUTINE MPINIT (X)	MP032823

C DECLARES BLANK COMMON (USED BY MP PACKAGE) AND	MP032825
C CALLS MPSET TO INITIALIZE PARAMETERS	MP032827
C THE AUGMENT DECLARATION	MP032829
C INITIALIZE MP	MP032831
C CAUSES A CALL TO MPINIT TO BE GENERATED.	MP032833
C *** ASSUMES OUTPUT UNIT 6, 43 DECIMAL PLACES,	MP032835
C *** 10 MP DIGITS, SPACE 296 WORDS. IF THE AUGMENT	MP032837
C *** DESCRIPTION DECK IS CHANGED THIS ROUTINE SHOULD	MP032839
C *** BE CHANGED ACCORDINGLY.	MP032841
COMMON B, T, M, LUN, MXR, R	MP032843
INTEGER B, T, X(1)	MP032845
C THE STATEMENTS	MP032847
INTEGER R(296)	MP032849
CALL MPSET (6, 43, 12, 296)	MP032851
C ARE A SPECIAL CASE OF	MP032853
C INTEGER R(MXR)	MP032855
C CALL MPSET (LUN, IDECPL, T+2, MXR)	MP032857
C WHERE LUN IS THE LOGICAL UNIT FOR OUTPUT,	MP032859
C IDECPL IS THE EQUIVALENT NUMBER OF DECIMAL PLACES REQUIRED,	MP032861
C T IS THE NUMBER OF MP DIGITS, AND	MP032863
C MXR IS THE SIZE OF THE WORKING AREA USED BY MP	MP032865
C (MXR = MAX (T*T+15*T+27, 14*T+156) IS SUFFICIENT).	MP032867
C TO CHANGE THE PRECISION, MODIFY THE DIMENSIONS IN THE	MP032869
C DECLARE STATEMENTS IN THE AUGMENT DESCRIPTION DECK -	MP032871
C THE DIMENSION FOR TYPE MULTIPLE SHOULD BE T+2 AND	MP032873
C FOR TYPE MULTIPAK SHOULD BE INT ((T+3)/2).	MP032875
C SEE COMMENTS IN ROUTINE MPSET FOR THE NUMBER OF MP	MP032877
C DIGITS REQUIRED TO GIVE THE EQUIVALENT OF ANY DESIRED	MP032879
C NUMBER OF DECIMAL PLACES.	MP032881
C *** ON SOME SYSTEMS A DECLARATION OF BLANK COMMON IN THE MAIN	MP032883
C *** PROGRAM MAY BE NECESSARY. IF SO, DECLARE	MP032885
C *** COMMON MPWORK(301)	MP032887
C *** OR, MORE GENERALLY,	MP032889
C *** COMMON MPWORK(MXR+5)	MP032891
C *** IN THE MAIN PROGRAM.	MP032893
RETURN	MP032895
END	MP032897
C \$\$ ***** MPIO *****	MP032921
SUBROUTINE MPIO (C, N, UNIT, IFORM, ERR)	MP032923
C IF UNIT .GT. 0 WRITES C(1), ... , C(N) IN FORMAT IFORM	MP032925
C IF UNIT .LE. 0 READS C(1), ... , C(N) IN FORMAT IFORM	MP032927
C IN BOTH CASES USES LOGICAL UNIT IABS(UNIT).	MP032929
C ERR IS RETURNED AS TRUE IF N NON-POSITIVE, OTHERWISE FALSE.	MP032931
C WE WOULD LIKE TO RETURN ERR AS TRUE IF READ/WRITE ERROR DETECTED,	MP032933
C BUT THIS CAN NOT BE DONE WITH ANSI STANDARD FORTRAN (1966).	MP032935
C *** UNIVAC ASCII FORTRAN (FTN 5R1AE) DOES NOT WORK IF IFORM	MP032937
C *** IS DECLARED WITH DIMENSION 1. MOST FORTRANS DO THOUGH.	MP032939
INTEGER C(N), UNIT, IFORM(20)	MP032941
LOGICAL ERR	MP032943
ERR = (N.LE.0)	MP032945
IF (ERR) RETURN	MP032947
IU = IABS(UNIT)	MP032949
IF (UNIT.GT.0) WRITE (IU, IFORM) C	MP032951
IF (UNIT.LE.0) READ (IU, IFORM) C	MP032953
RETURN	MP032955

END	MP032957
C \$\$ ***** MPKSTR *****	
SUBROUTINE MPKSTR (X, Y)	MP032961
C SETS Y = X FOR PACKED MP NUMBERS X AND Y.	MP032963
C ASSUMES SAME PACKED FORMAT AS MPPACK AND MPUNPK.	MP032965
COMMON B, T, M, LUN, MXR, R	MP032967
INTEGER B, T, R(1), X(2), Y(2)	MP032969
Y(2) = X(2)	MP032972
C CHECK FOR ZERO	MP032973
IF (Y(2).EQ.0) RETURN	MP032975
C HERE X NONZERO SO MOVE PACKED NUMBER	MP032977
N = (T+3)/2	MP032979
DO 10 I = 1, N	MP032981
10 Y(I) = X(I)	MP032983
RETURN	MP032985
END	MP032987
	MP032989
C \$\$ ***** MPLE *****	
LOGICAL FUNCTION MPLE (X, Y)	MP033001
C RETURNS LOGICAL VALUE OF (X .LE. Y) FOR MP X AND Y.	MP033003
INTEGER X(1), Y(1)	MP033005
MPLE = (MPCOMP(X,Y) .LE. 0)	MP033007
RETURN	MP033009
END	MP033011
	MP033013
C \$\$ ***** MPLT *****	
LOGICAL FUNCTION MPLT (X, Y)	MP037281
C RETURNS LOGICAL VALUE OF (X .LT. Y) FOR MP X AND Y.	MP037283
INTEGER X(1), Y(1)	MP037285
MPLT = (MPCOMP(X,Y) .LT. 0)	MP037287
RETURN	MP037289
END	MP037291
	MP037293
C \$\$ ***** MPMEXA *****	
FUNCTION MPMEXA (X)	MP038051
C RETURNS THE MAXIMUM ALLOWABLE EXPONENT OF MP NUMBERS (THE THIRD	MP038053
C WORD OF COMMON). X IS A DUMMY MP ARGUMENT.	MP038055
COMMON B, T, M, LUN, MXR, R	MP038057
INTEGER B, T, R(1), X(1)	MP038059
MPMEXA = M	MP038061
RETURN	MP038063
END	MP038065
	MP038067
C \$\$ ***** MPMEXB *****	
SUBROUTINE MPMEXB (I, X)	MP038071
C SETS THE MAXIMUM ALLOWABLE EXPONENT OF MP NUMBERS (I.E. THE	MP038073
C THIRD WORD OF COMMON) TO I.	MP038075
C I SHOULD BE GREATER THAN T, AND 4*I SHOULD BE REPRESENTABLE	MP038077
C AS A SINGLE-PRECISION INTEGER.	MP038079
C X IS A DUMMY MP ARGUMENT (AUGMENT EXPECTS ONE).	MP038081
COMMON B, T, M, LUN, MXR, R	MP038083
INTEGER B, T, R(1), X(1)	MP038085
M = I	MP038087
C CHECK LEGALITY OF M. IF TOO LARGE, 4*M MAY OVERFLOW AND TEST .LE. 0	MP038089
IF ((M.GT.T).AND.((4*M).GT.0)) RETURN	MP038091
	MP038093

WRITE (LUN, 10)	MP038095
10 FORMAT (44H *** ATTEMPT TO SET ILLEGAL MAXIMUM EXPONENT,	MP038097
\$ 22H IN CALL TO MPMEXB ***)	MP038099
CALL MPERR	MP038101
RETURN	MP038103
END	MP038105
C \$\$ ***** MPNE *****	MP040461
LOGICAL FUNCTION MPNE (X, Y)	MP040463
C RETURNS LOGICAL VALUE OF (X .NE. Y) FOR MP X AND Y.	MP040465
INTEGER X(1), Y(1)	MP040467
MPNE = (MPCOMP(X,Y) .NE. 0)	MP040469
RETURN	MP040471
END	MP040473
C \$\$ ***** MPOUTF *****	MP041781
SUBROUTINE MPOUTF (X, P, N, IFORM, ERR)	MP041783
C WRITES MP NUMBER X ON LOGICAL UNIT LUN (FOURTH WORD OF COMMON)	MP041785
C IN FORMAT IFORM AFTER CONVERTING TO FP.N DECIMAL REPRESENTATION	MP041787
C USING ROUTINE MPOUT. FOR FURTHER DETAILS SEE COMMENTS IN MPOUT.	MP041789
C IFORM SHOULD CONTAIN A FORMAT WHICH ALLOWS FOR OUTPUT OF P	MP041791
C WORDS IN A1 FORMAT, PLUS ANY DESIRED HEADINGS, SPACING ETC.	MP041793
C E.G. 24H(8H1HEADING/(11X,100A1))	MP041795
C ERR RETURNED AS TRUE IF P NOT POSITIVE, OTHERWISE FALSE.	MP041797
C SPACE REQUIRED 3T+P+11 WORDS.	MP041799
COMMON B, T, M, LUN, MXR, R	MP041801
INTEGER B, T, R(1), X(1), IFORM(1), P	MP041803
LOGICAL ERR	MP041805
ERR = .TRUE.	MP041807
C RETURN WITH ERROR FLAG SET IF OUTPUT FIELD WIDTH P NOT POSITIVE	MP041809
IF (P.LE.0) RETURN	MP041811
C CHECK THAT ENOUGH SPACE IS AVAILABLE	MP041813
CALL MPCHK (3, P+11)	MP041815
I2 = 3*T + 12	MP041817
C CONVERT X TO DECIMAL FORM	MP041819
CALL MPOUT (X, R(I2), P, N)	MP041821
C AND WRITE ON UNIT LUN WITH FORMAT IFORM	MP041823
CALL MPIO (R(I2), P, LUN, IFORM, ERR)	MP041825
RETURN	MP041827
END	MP041829
C \$\$ ***** MPSIGA *****	MP048741
FUNCTION MPSIGA (X)	MP048743
C RETURNS SIGN OF MP NUMBER X	MP048745
INTEGER X(1)	MP048747
MPSIGA = X(1)	MP048749
RETURN	MP048751
END	MP048753
C \$\$ ***** MPSIGB *****	MP048761
SUBROUTINE MPSIGB (I, X)	MP048763
C SETS SIGN OF MP NUMBER X TO I.	MP048765
C I SHOULD BE 0, +1 OR -1.	MP048767
C EXPONENT AND DIGITS OF X ARE UNCHANGED,	MP048769
C BUT RESULT MUST BE A VALID MP NUMBER.	MP048771
COMMON B, T, M, LUN, MXR, R	MP048773

INTEGER B, T, R(1), X(3)	MP048775
X(1) = I	MP048777
C CHECK FOR VALID SIGN	MP048779
IF (IABS(I).LE.1) GO TO 20	MP048781
WRITE (LUN, 10)	MP048783
10 FORMAT (39H *** INVALID SIGN IN CALL TO MPSIGB ***)	MP048785
GO TO 40	MP048787
C RETURN IF X ZERO	MP048789
20 IF (I.EQ.0) RETURN	MP048791
C CHECK FOR VALID EXPONENT AND LEADING DIGIT	MP048793
IF ((IABS(X(2)).LE.M).AND.(X(3).GT.0).AND.(X(3).LT.B)) RETURN	MP048795
WRITE (LUN, 30)	MP048797
30 FORMAT (48H *** X NOT VALID MP NUMBER IN CALL TO MPSIGB ***)	MP048799
40 CALL MPERR	MP048801
X(1) = 0	MP048803
RETURN	MP048805
END	MP048807
C \$\$ ***** MPUPK *****	MP052341
SUBROUTINE MPUPK (SOURCE, DEST, LDEST, LFIELD)	MP052343
C	MP052345
C *****	MP052347
C *** MACHINE DEPENDENT ***	MP052349
C *****	MP052351
C	MP052353
C MACHINE-DEPENDENT STATEMENTS ARE SURROUNDED BY C *** LINES	MP052355
C ***	MP052357
C THIS IS UNIVAC 1100, FORTRAN V VERSION.	MP052359
C ***	MP052361
C THIS SUBROUTINE UNPACKS A PACKED HOLLERITH STRING (SOURCE)	MP052363
C PLACING ONE CHARACTER PER WORD IN THE ARRAY DEST (AS IF READ IN	MP052365
C A1 FORMAT). IT CONTINUES UNPACKING UNTIL IT FINDS A SENTINEL (\$)	MP052367
C OR UNTIL IT FINDS A COMPILER GENERATED SENTINEL (IF SO	MP052369
C IMPLEMENTED) OR UNTIL IT HAS FILLED LDEST WORDS OF THE	MP052371
C ARRAY DEST. THE LENGTH OF THE UNPACKED STRING IS RETURNED	MP052373
C IN LFIELD. THUS 0 .LE. LFIELD .LE. LDEST.	MP052375
INTEGER SOURCE(1), DEST(1), BLANKS, TEMP	MP052377
DATA BLANKS /1H /, IST /1H\$/	MP052379
C NK IS THE NUMBER OF CHARACTERS PER WORD	MP052381
C AND ISTC IS THE COMPILER-GENERATED SENTINEL (IF ANY)	MP052383
C ***	MP052385
DATA NK /6/, ISTC /0/	MP052387
C ***	MP052389
TEMP = BLANKS	MP052391
LD = LDEST	MP052393
LFIELD = 0	MP052395
IF (LD.LE.0) RETURN	MP052397
DO 10 K = 1, LD	MP052399
I = LFIELD/NK + 1	MP052401
C GET NEXT WORD (CONTAINING NK CHARACTERS) AND	MP052403
C CHECK FOR COMPILER-GENERATED END-OF-STRING SENTINEL	MP052405
IF (SOURCE(I) .EQ. ISTC) RETURN	MP052407
C MOVE (MOD(LFIELD,NK)+1)-TH CHARACTER OF SOURCE(I) TO	MP052409
C FIRST (I.E. LEFTMOST) CHARACTER POSITION OF TEMP	MP052411
C ***	MP052413
FLD (0, 6, TEMP) = FLD (6*MOD(LFIELD,6), 6, SOURCE(I))	MP052415

<pre> C *** C CHECK FOR END-OF-STRING SENTINEL IF (TEMP .EQ. IST) RETURN LFIELD = K 10 DEST(K) = TEMP RETURN END SUBROUTINE MPUPK (SOURCE, DEST, LDEST, LFIELD) C C ***** C *** MACHINE DEPENDENT *** C ***** C C MACHINE-DEPENDENT STATEMENTS ARE SURROUNDED BY C *** LINES C *** C THIS IS UNIVAC 1100, ASCII FORTRAN VERSION. C *** C THIS SUBROUTINE UNPACKS A PACKED HOLLERITH STRING (SOURCE) C PLACING ONE CHARACTER PER WORD IN THE ARRAY DEST (AS IF READ IN C A1 FORMAT). IT CONTINUES UNPACKING UNTIL IT FINDS A SENTINEL (\$) C OR UNTIL IT FINDS A COMPILER GENERATED SENTINEL (IF SO C IMPLEMENTED) OR UNTIL IT HAS FILLED LDEST WORDS OF THE C ARRAY DEST. THE LENGTH OF THE UNPACKED STRING IS RETURNED C IN LFIELD. THUS 0 .LE. LFIELD .LE. LDEST. INTEGER SOURCE(1), DEST(1), BLANKS, TEMP DATA BLANKS /1H /, IST /1H\$/ C NK IS THE NUMBER OF CHARACTERS PER WORD C AND ISTC IS THE COMPILER-GENERATED SENTINEL (IF ANY) C *** DATA NK /4/, ISTC /0/ C *** TEMP = BLANKS LD = LDEST LFIELD = 0 IF (LD.LE.0) RETURN DO 10 K = 1, LD I = LFIELD/NK + 1 C GET NEXT WORD (CONTAINING NK CHARACTERS) AND C CHECK FOR COMPILER-GENERATED END-OF-STRING SENTINEL IF (SOURCE(I) .EQ. ISTC) RETURN C MOVE (MOD(LFIELD,NK)+1)-TH CHARACTER OF SOURCE(I) TO C FIRST (I.E. LEFTMOST) CHARACTER POSITION OF TEMP C *** BITS (TEMP, 1, 9) = BITS (SOURCE(I), 9*MOD(LFIELD,4)+1, 9) C *** C CHECK FOR END-OF-STRING SENTINEL IF (TEMP .EQ. IST) RETURN LFIELD = K 10 DEST(K) = TEMP RETURN END C \$\$ ***** MPUPK ***** SUBROUTINE MPUPK (SOURCE, DEST, LDEST, LFIELD) C </pre>	<pre> MP052417 MP052419 MP052421 MP052423 MP052425 MP052427 MP052429 MP052341 MP052343 MP052345 </pre>
---	---

C	*****	MP052347
C	*** MACHINE DEPENDENT ***	MP052349
C	*****	MP052351
C		MP052353
C	MACHINE-DEPENDENT STATEMENTS ARE SURROUNDED BY C *** LINES	MP052355
C	***	MP052357
C	THIS IS IBM 360 FORTRAN G OR H VERSION	
C	***	MP052361
C	THIS SUBROUTINE UNPACKS A PACKED HOLLERITH STRING (SOURCE)	MP052363
C	PLACING ONE CHARACTER PER WORD IN THE ARRAY DEST (AS IF READ IN	MP052365
C	A1 FORMAT). IT CONTINUES UNPACKING UNTIL IT FINDS A SENTINEL (\$)	MP052367
C	OR UNTIL IT FINDS A COMPILER GENERATED SENTINEL (IF SO	MP052369
C	IMPLEMENTED) OR UNTIL IT HAS FILLED LDEST WORDS OF THE	MP052371
C	ARRAY DEST. THE LENGTH OF THE UNPACKED STRING IS RETURNED	MP052373
C	IN LFIELD. THUS 0 .LE. LFIELD .LE. LDEST.	MP052375
	INTEGER DEST(1), BLANKS, TEMP	
C	***	
	LOGICAL*1 SOURCE(1), TC(4)	
	EQUIVALENCE (TC, TEMP)	
C	***	
	DATA BLANKS /1H /, IST /1H\$/	MP052379
C	NK IS THE NUMBER OF CHARACTERS PER WORD	MP052381
C	AND ISTC IS THE COMPILER-GENERATED SENTINEL (IF ANY)	MP052383
C	***	MP052385
	DATA NK /4/, ISTC /0/	
C	***	MP052389
	TEMP = BLANKS	MP052391
	LD = LDEST	MP052393
	LFIELD = 0	MP052395
	IF (LD.LE.0) RETURN	MP052397
	DO 10 K = 1, LD	MP052399
C	***	MP052413
	TC(1) = SOURCE (K)	
C	***	MP052417
C	CHECK FOR END-OF-STRING SENTINEL	MP052419
	IF (TEMP .EQ. IST) RETURN	MP052421
	LFIELD = K	MP052423
	10 DEST(K) = TEMP	MP052425
	RETURN	MP052427
	END	MP052429

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 1868	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) AN AUGMENT INTERFACE FOR BRENT'S MULTIPLE PRECISION ARITHMETIC PACKAGE		5. TYPE OF REPORT & PERIOD COVERED Summary Report - no specific reporting period
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Richard P. Brent, Judith A. Hooper, J. M. Yohe		8. CONTRACT OR GRANT NUMBER(s) DAAG29-75-C-0024
9. PERFORMING ORGANIZATION NAME AND ADDRESS Mathematics Research Center, University of 610 Walnut Street Madison, Wisconsin 53706		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 8 Computer Science
11. CONTROLLING OFFICE NAME AND ADDRESS U. S. Army Research Office P. O. Box 12211 Research Triangle Park, North Carolina 27709		12. REPORT DATE August 1978
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		13. NUMBER OF PAGES 22
		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Arithmetic, multiple precision, extended precision, floating point, portable software, software package, precompiler interface, AUGMENT interface.		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) We describe the procedure required to interface the FORTRAN multiple precision package of Richard P. Brent (as described in ACM Transactions on Mathematical Software, March, 1978) with the AUGMENT precompiler for FORTRAN. We also indicate the method of using the multiple precision arithmetic package in conjunction with AUGMENT.		