15. データタイプチェックに関する一考察

東京大学　理学部　後藤英一
理化学研究所　井田哲雄

1. 序論

数値計算を主体とするプログラム（FORTRANで書かれているとする）を書いたが、それに使ったアルゴリズムは正しいと信ぜられるにもかかわらず、数値のオーバーフロー、又は精度の不足（Double Precisionでも）のためにうまく働かないという事態に直面したものとしよう。現存の大多数のプログラミング・システムはこのような事態を自動的に処理する機能を備えていないので、問題はかなり深刻である。そこで、計算機関係の専門家に相談すれば、次のような回答とコメントが得られるであろう。

〔I〕数値計算の専門家：「もっと桁落ちの少ないアルゴリズム、数値の大きさが限定されるようなアルゴリズムを使ったらいいでしょう。」

〔II〕経験豊富な応用プログラマー：「まず算術演算をすべてサブルーチンの呼び出しに変更する。例えば、"Z＝X＋Y"は"CALL BIGADD(Z,X,Y)"に変える。そして変数は適当な配列に変更する。例えば、

DIMENSION X(6), Y(6), Z(6)

とする。こうすれば、大指数型浮動小数点数，超高精度浮動小数点数，超多倍長整数などを取扱うのに必要なサブルーチンのいくつかを自分で書きさえすれば問題は解決される。このようなサブルーチンのパッケージがライブラリーにあれば、これを利用すればよい。またこのようなサブルーチンを書く上に最近の文献としてはWyatt達の論文〔1〕が参考になるでしょう。」

〔III〕問題が超多倍長整数にあれば，数式処理の専門家は次のようにコメントするであろう。「最近の数式処理システムは、整数をすべて任意多倍長整数として正しく取扱う。（例えば文献〔2〕,〔3〕参照）数値計算は数式処理の特別な場合であるから、数式処理システム（MACSYMA, REDUCEなど）が使えれば問題はないはずである。」

〔IV〕問題が大指数，超高精度浮動小数点数にあれば，特にMITの数式処理システムMACSYMAの利用法に精通している人ならば、次のようにコメントするであろう。「MACSYMAのBIGFLOAT機能を使えば問題はないはずである。（文献〔4〕参照）

〔V〕コンパイラ作成の専門家：「この種の問題を完全に自動的に処置するような新しいコンパイラを書く上に実行速度を多少犠牲にすれば原理上の困難は全くないはずである。」

本稿では，まず次節でこの問題を純ソフト的に解決する諸方法についてまず論じ、次いでこの問題を合理的かつ高能率で取扱うのに適する"TRAPNUM"と呼ぶアーキテクチュアを提案する。さらに"TRAPNUM"法をサブルーチン呼出し方式の改善に利用する可能性についてもふれる。

2. "大きな数"を取扱うソフトウェア

まず前節〔I〕の解決法については、桁落ちの少ないアルゴリズム，数値が無闇に大きくならないアルゴリズムに変更できる例は数値解析の教科書にも多数見られ

るが，このような変更が何時でも容易にできるとは限らない。一例として，1975年のScientific American 4月号をにぎわしたGardnerの"四月馬鹿"記事がある。その一つに「$e^{\sqrt{163}\pi}$が整数であることが証明された」というのがある。これを普通のFORTRANで

X = EXP (SQRT (163.0) * 3.141592)

によって計算したのでは，結果が0.262537E+18のように整数部分が十進18桁の数になることが分るだけで，"四月馬鹿"であることは見抜けない。この数値は実は小数点以下9が12個続いて13桁目にはじめて9ではなくて2が現れるという非常に整数に近い数なのである。従って，少なくとも十進31桁の精度で計算する必要がある。この結果が普通のFORTRANで何の工夫もなくすぐに出せるようであったならば，"四月馬鹿"としては成立しなかったであろう。この問題は"四月馬鹿"よりも遙かに深刻で，現在の数値計算用プログラミング言語の欠陥に対する痛烈な批判と受けとるべきであろう。

前節(II)，(III)，(IV)の解決法は，FORTRANで一回書いてしまったプログラムを全部書き換える必要がある点に問題がある。この書き換えはアルゴリズムの本質を変更するのではないので，全く機械的な作業となる。このような機械的作業は，本来計算機で行えるはずである。

この点に関し，Wyatt達のSUPER PRECISIONシステムでは(II)の算術演算のサブルーチン展開法を採用しているが，この展開はPrecompilerによって自動的に行うようになっている。

MACSYMAやREDUCE（文献(2,3)参照）などの数式処理システムは非常に強力な記号処理能力も備えているので，FORTRANをLISPまたはMACSYMA語，またはREDUCE語（これらの数式処理システムはLISPをHost言語として書かれた数式処理言語システムである）に翻訳するTranslatorプログラムを書くことは，その気になればそう難しいことではない。筆者達は学部4年の小野君と共にFORTRANからHLISP（文献(6)参照）へのTranslatorを3人月程度で作成することができた。ただし本稿で論ずる"大きな数"を取扱う機能は，まだ含まれていない。現在はこのTranslatorを言語FLATS（文献(7)参照）のFORTRANコンパイラに改良し，"大きな数"の処理機能も含める作業を行っている。

筆者達は最近ハッシング・ハードウェアの動作の確率論的解析を行ったが，それに伴う数値計算のFORTRANプログラムで，浮動小数点指数オーバーフローの問題に遭遇した。上記のFLATSコンパイラが未完のため困惑したが，FACOM 230-60/75, FORTRAN-C/Dに TYPE宣言文機能があることを知り，これを利用して問題を解決した。この方法の概要は御参考までに付録に記載しておくが，一口で言えば，前節(II)において，TYPE文で宣言した変数が関与する算術演算をサブルーチン呼出しに展開することをメーカー提供のコンパイラが行ってくれるというもので，ユーザーは展開サブルーチンをいくつか書きさえすればよい。結果的に見れば，WyattのSUPER PRECISIONシステムと殆んど同一の方法で問題を解決したことになるが，Wyatt達のPrecompilerの部分はFORTRAN本来のコンパイラが実行してくれるので，この部分は作成する必要はない。

TYPE宣言文機能とHLISP-REDUCEシステムにおける任意多倍長整数演算機能の使用経験及びWyatt達のSUPER PRECISION Package（文献(1)参照）とFatemanのMACSYMA-BIGFLOAT Package（文献(4)参照）の仕様を机上で検討した結果から，ユーザーに

とって使い易い機能重視型（実行能率の向上の問題は次節に論ずる）の"大きな数（BIGNUM）"用のソフトウェア・システムは次のようにあることが望ましいと考える。

5. システム基本組込み機能（整数と浮動小数点数）

5.1 整数. Integerと宣言（FORTRANのI-M暗黙宣言を含む）された変数は、常に任意多倍長整数として取扱い記憶容量の許す限りどんなに長い整数に関する算術演算でも正しく実行する。これはMACSYMA, REDUCEなどの進んだ数式処理システムでは常識であるが、Wyatt等のSUPER PRECISIONシステムの機能は考慮されていない。

5.2 指数. 浮動小数点数の指数は5.1の意味での"本当"の整数とし、浮動小数点の指数のオーバー・フローとアンダー・フローは決して起らないようにする。BIGFLOAT（文献(4)参照）はこのような取扱いになっているが、SUPER PRECISION（文献(1)参照）では指数のオーバー・フローとアンダー・フローの問題は全く考慮されていない。

5.3 整数と浮動小数点数の混用. これはコンパイラル時ではなく、実行時にも許されるべきである。BIGFLOAT（文献(4)参照）ではこれは許されるが、SUPER PRECISION（文献(1)参照）ではこれはコンパイル時に全部決まっていなくてはならない。

5.4 精度指定とASW（Arithmetic Status Word） 浮動小数点数の演算結果の精度指定は、PSW（Program Status word）の一部分に含まれるとみなされるASWによって行う。BIGFLOAT（文献(4)参照）ではこれを精度指定大局的変数（Global Variable）と呼んでいるが、システム・ソフト的観点からはASWと呼ぶ方がふさわしい。SUPER PRECISION（文献(1)参照）とTYPE宣言法（付録）では精度は変数の型宣言で静的に決めてコンパイルするが、BIGFLOAT（文献(4)参照）のように動的に決める方が使い易い。

5.5 ASWによる誤り検出のOption指定. 零除算などの異常事態に関する処置のOption指定は、ASWにタグをつけ階層的ブロック構造をもって変更できるようにする。

6. ソフトウェア型指定機能 前項のシステム基本組込み機能には有理数，複素数，有理複素数（Gauss整数），精度指数付数値等々の意味に変数と算術演算子（+, −, ×, /, **の他にソフトウェアでその意味を指定する演算子//, %/%などがあれば、/は浮動小数点割算、//は整数商、%/%は有理数商等に使いわけられて一層便利であろう）を自由にソフトウェア的に指定できるようにする。TYPE宣言機能とSUPER PRECISIONのPrecompiler（文献(1)参照）でもこれは可能であり，またMACSYMAとBIGFLOAT（文献(4)参照）システムでもこれは原理上可能なはずであるが，合理的で使い易いシステムにするには、今後さらに研究を進める必要があろう。

以上に述べたような機能を持つ拡張FORTRAN（ALGOL 60等の数値計算指向言語でも同様であろう）コンパイラをポータブルの型（例えば基本的なFORTRANで書く）に作成することは、純機能的には比較的容易であろうが，データの型検査を動的に行う必要があるため実行速度は通常のFORTRANより1桁か2桁落ちてしまうであろう。しかし，言語の使い易さとその効用を純ソフト的に実験してみることは，大いに意味のあることと信ぜられる。

前例の"四月馬鹿"の問題は拡張FORTRAN（FLATS）では、次のように書けばよ

いようになる。

```
1       PRECISION/31/DECIMALS
2       PI = 4 * ATAN (1)
3       X = EXP(SQRT(163)*PI)
4       WRITE (6,5) X
5       FORMAT (1HO,F32.13)
6       STOP
7       END
```

結果は、262537412640768743.999999999992。

ここで行1の文により ASW が浮動小数点を31桁以上でおめられるようにセットされる。（このプログラムが副プログラムであれば、前の ASW 精度指定値はスタックに保存される） 標準関数 ATAN, SQRT, EXP はこの ASW 精度値に従って31桁正しく計算される。円周率の値に関してはこれをN桁の精度でおめる組込み関数、PI(N)を作った方がスマートかもしれない。（BIGFLOATにはこのような関数がある）またこのささやかな紙上実験からも"大きな数"の出力には FORTRAN の FORMAT だけでは不充分で、FORMAT指定子、例えば N=3 として F32.13 BLANK(3), F32.13 COMMA(3) などの新構文を導入したくなる。

3. TRAPNUM法

前節に示したような"BIGNUM大きな数"に関する操作を高速度で能率よく実行するには、Iliffeの仮想的な計算機 BLM（文献(8)参照）のようなタグ・マシンを使えばよい。しかしすべての数値データ語に目印のタグをつけてハードウェアによって実行時にデータ型検査を高速で行う方法の欠点は、語中のビット利用効率の低下にある。例えば1語32bitの計算機にnbitをタグに使うとnbitの損失をきねいて、1倍長浮動小数点数の指数か仮数（mantissa）のbitをn個減らさなくてはならなくなる。この問題はタグbitではなく、特別な指数値をタグと見るようにすれば解決される。今、6通りのタグが必要とすると、$2^2<6<2^3$ であるからタグbit法では3bitのbit損失が起る。ところが、IBM 360 のように7bitの指数 e ($-64\leq e\leq 63$) がある場合に、$e=-64,-63,-62,+62,+63$ を特別なタグ付き語と見なし、$-61\leq e\leq +61$ は1倍長小指数型浮動小数点とみることとすれば、128通り使える指数が123=128-5通りに減ずるだけであるから、bit損失は $\log_2 123/128 = -0.055$ bit にしか相当しない。前節の"BIGNUM"の処理にはタグを次のように使用すればよい。

e	名称	意味
63	BADNUM	演算過程中の零除算等々の不合理な結果。
62	BIGFLOAT	仮数部（24bit）は大指数型又は多精度型浮動小数点数データへのポインタ。
-62	SINT	仮数部は短整数（$|I|<2^{24}$）を表す。
-63	BIGINT	仮数部は多倍長整数（$|I|\geq 2^{24}$）データへのポインタ。
-64	TRAPNUM	仮数部はソフトウェア的に定義された データ型へのポインタで、ソフトウェア割出し（"TRAP"）によって処理する。

BIGNUM と BIGINT もソフトウェアで処理すれば、これらも一種の TRAPNUM (Trap Number割り出し型数) と見なせる。この方法により短整数（SINT）と1倍長小指数

点数(含FLOAT)に関する演算は従来の命令と同一の速度で働くことになる。統計的には整数値としてはSINTがBIGINTより圧倒的に多く，また小指数($|e|\leq 61$)は大指数($|e|>61$)よりも圧倒的に多く現れるであろうから平均実行速度は前節の純ソフトによる方法より著しく向上するはずである。BIGINT, BIGFLOATとポインタの先のデータ構造は一般ユーザーからは見えない(見えてはならない)部分であり，この部分をどのように速度と記憶容量に関し能率よく作成するかは，システム作成者の腕の見せどころであり，それには記憶領域の動的管理が必要である。普通のFORTRANはコンパイル時に記憶領域の割当てが可能という点が長所ともされているが，FORTRAN誕生当時に比較して，記憶領域の動的管理技術は非常に進歩しているので，今日ではこれを長所と数える必要は全く見当らない。

本節までに示した"大きな数"の処理方式に従えば，異種システム間の数値の意味の互換性の欠除の問題は完全に解決することにも注意されたい。因みに通常のFORTRANの"整数"には機種によって16bitから60bit固定長までの変種があり，"実数"の大きさの範囲も$2^{\pm 7}$のもの(例えばACOS)から$2^{\pm 11}$のもの(例えばCDC)まであって，これに起因する互換性の欠除にはユーザーは泣かされ続けてきたのである。

4. 結論

電子計算機の最も古典的な応用分野である数値計算に関し，今日広く普及している"高級"プログラミング・システムが長整数，大指数高精度浮動小数点数などの"大きな数"の演算に関し今なお全く無力の状態にあるというのは驚くべき事実である。このためユーザーはこの種の演算のために，独自の工夫をすることを強いられてきた。因みにFORTRANの誕生以来この点に関するFORTRANの改良と言えば，DOUBLE PRECISION(又はQUADRUPLE)宣言の採用しか見当らない。

このような事態が生じた原因の一つに，ユーザーの要求とコンパイラ作成者と計算機アーキテクチュア設計者相互間の情報交流の不足があげられる。割込み，記憶保護，I/Oチャンネル等々の機構に欠けるところがあれば，満足なOSは作成不可能であるからOS作成者とアーキテクチュア設計者の間には正のフィードバックが見られる。これに対し超大型機からマイクロコンまで，どのようなアーキテクチュアを天下り的に与えられた場合にも，それに順応してコンパイラが書かれてきて，フィードバックは少なかったように見受けられる。

本稿のTRUPNUM機構は，ハードウェアのごくわずかの変更によっても，"大きな数"の演算が大中に合理化できることを例示したまでのもので，ここでこの機構が最良であると主張するものではない。

計算機システムの合理的設計は諸概念と術語の合理化から出発すべきと考える。その第一原則として「他の分野ですでに確立している概念と術語の意味を不用意に変えるな」をかかげたい。この原則に反する第1の例としては，有史以前の数学から確立している"整数 Integer"の"Machine Dependent"な固定長固定小数点数の意味への転用があげられる。"INTEGER"と宣言された変数は"任意多倍長整数"の意味以外に解釈されるべきではない。"実数 Real"も不当で"浮動(小数点)数 FLOATNUM"と呼ぶべきであろう。

第2の例は等号"="の代入演算子への流用($N=N+1$等)で，これは":=","←"ならよい。

第3の例としては関数表記号法と引数引渡し法の混乱があげられる。引数引き渡し法には少くとも"Call by reference", "Call by value", "Call by result", "Call by value-result", "Call by name"の5種がある。(文献[9]参照) 関数表記法F(X,Y,Z)は殆んどすべての高級言語に利用されているが、数学の常識からすれば、この表記法は"Call by value"のみを意味すべきである。他の引数引渡し法は呼ばれた子側で勝手に決めるのではなく、呼ぶ親側のシンタックスに従って"top-down"的に決める方が、プログラムのmodularityをよくする意味からも合理的であろう。例えば、F(X:=,Y,/Z/)とすればXは"Call by reference", Yは"Call by value", Zは"Call by name"とするのである。この引数引渡し法を現在の計算機で合理的にコンパイルするには、大局的コンパイラが必要で、副プログラム単位のコンパイルは実行速度を重視すると難しい。この問題の解決法の一例としては、前節のタグ付ポインター語に次の3種を追加すればよい。

e	名称	意味
61	ROP	Read Only Pointerによる自動間接アドレス。
-61	RWP	Read Write Pointerによる自動間接アドレス。
-60	NP	Name Pointer "Call by name"のテキストまたはそれをコンパイルしたコードへのポインタ。

親のF(X:=,Y,/Z/)はXをRWPで, Yをvalueで, ZをNPで子に引渡せばよい。なお"Call by value-result"の場合にはvalueをstackに積んで引渡し、親に戻ったときにresultの処置をするようにコンパイルする。なお子側のF(X,Y,Z)の仮引数はそれぞれ"given by reference", "given by value", "given by name"で引渡しを受けたと言ったらよいのではあるまいか。

以上で電子計算機の最も古いデータ型である数値に関してすら、その表現と処理方式には多大な問題が残されていることが明らかになった。配列に始まるより新しい各種のデータ型の定義、表現と処理方式の再考は今後に残された課題である。

References

[1] Wyatt, W. T., Lozier, P. W. and Orser, P. J.:
"A Portable Extended Precision Arithmetic Package and Library With Fortran Precompiler",
ACM Trans. Math. Software, vol. 2 (1976) pp. 209-231.

[2] Hearn, A. C.: "REDUCE-2 User's Mannual",
University of Utah, Salt Lake City, Utah, March 1973.

[3] The Mathlab Group "MACSYMA Mannual", MIT, Cambridge Mass. 1974, 1976.

[4] Fateman, R. J.,: "The MACSYMA 'Big-Floating-Point' Arithmetic System"
in Porceeding of ACM-SYMSAC '76,
Yorktown Heights NY, Aug. 1976.

[5] Gardner, M.: "Mathematical Games", in the April Issue of Scientific American 1975.

[6] 後藤英一: "LISP入門" bit 1974年1月－75年2月号.

[7] 後藤英一: "言語FLATSとSP試論" 数理科学 1976年7月－8月号.

[8] Iliffe, J. K.: "Basic Machine Principles" (2nd ed.), MacDonald, London, 1971.

[9] Gries, D.,: "Compiler Construction for Digital Computers", John Wiley & Sons, Inc.
New York, 1971

付録　ＴＹＰＥ文の機能とその応用について

拡張ＦＯＲＴＲＡＮにおけるＴＹＰＥ文は，プログラマーにＦＯＲＴＲＡＮ文法に組みこまれたタイプ以外のデータタイプをもった変数の使用を許すものである。この機能はＦＡＣＯＭ２３０－７５/６０のＦＯＲＴＲＡＮＣ，Ｄコンパイラーが許すＦＯＲＴＲＡＮ文法仕様に取りいれられている。文献［ＦＡＣＯＭ２３０－６０　ＦＯＲＴＲＡＮ文法編　ＳＰ－０６１－４－５］によれば，ＴＹＰＥ文は次のように記述される。すなわち

ＴＹＰＥ　　Ｓ＊Ｗ　l_1，l_2，・・・

と書き，ここでＳは特殊型名で１～３個の英数字の文字列からなり，その最初の文字は英字でなければならない。l_1，l_2，・・・は＜要素名＞（：＝＜配列要素＞|＜変数名＞）である。Ｗはこのようにして導入された＜要素＞が占める語数を指定するもので１～８の指定数である。言いかえれば，ＴＹＰＥ文は新しいタイプの定義，およびそのタイプを持った変数の宣言の２つの機能を行う宣言文である。このようにして宣言された変数名に付する演算および，算術ＩＦ文はコンパイラーによって検出され，対応する乗除算等の機械語命令列が生成される代りにタイプ名，演算の種類および被演算項のタイプによって決定される名前をもったサブルーチンへの呼び出しコードが生成される。サブルーチンの名前の命名規則は以下のようである。

演算の場合：サブルーチン名＝＜特殊型名＞＜演算頭文字＞＜第１被演算項の型頭文字＞＜第２被演算項の型頭文字＞，(Unaryの時は第１被演算項の型頭文字と同じ)

算術ＩＦ文の場合：サブルーチン名＝＜特殊型名＞Ｉ＜算術ＩＦ文中に現われる式のもつ型頭文字＞＜算術ＩＦ文中に現われる式のもつ型頭文字＞

ここで演算頭文字は

Ａ ── Addition，Ｓ ── Subtraction　　Ｍ ── Multiplication，Ｐ ── Power
Ｄ ── Division，Ｔ ── Transfer　　Ｉ ── IF(Arithmetic)，Ｕ ── Unary negate

で与えられ，型頭文字は

Ｉ ── 整数型，Ｒ－実数型，　　Ｄ ── 倍精度実数型，Ｃ－複素数型，
Ｂ ── 倍精度複素数型，Ｌ－論理型，１ ── 第１番目に定義された特殊型，
２ ── 第２番目に定義された特殊型，３ ── 第３番目に定義された特殊型，

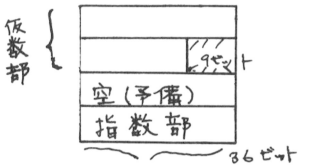
で与えられる。

サブルーチンへの引数は，第１引数が，結果が代入される変数名，第２引数は第１被演算項名，第３引数は第２被演算項名である。但し，Unaryの時は第３引数はない。算術ＩＦ文の時は第１引数が式の値をもつ変数名，第２，３，４は式の値が各々負，ゼロ，正の時の飛び出し番地である。

このようにして、ＴＹＰＥで宣言されたタイプを有する変数に対する上記演算は，ユニークな名前をもったサブルーチン呼び出しに変るため，プログラマーはこのサブルーチンの本体を，コーディングすることにより，特殊タイプに対する演算の意味を自由に与えることができる。

具体例として次にプログラムを示す。プログラム（Ａ）は係数がある漸化式で表現される多項式を計算するものであるが，文番号２，３のタイプ文の削除，及びBIGABSを通常の絶対値計算サブルーチンで置きかえると，指数部のアンダーフロを越し，正しい答を与えない。このため，タイプＢＩＧが導入された。このタイプの変数の値は指数部分が大きく，仮数部分が二倍長で表現される。（但しこ

の事実はプログラム(A)からは見えない。) サブルーチンCOMPFをコンパイルすると，タイプBIGの変数の演算に関連して，BIGA11，BIGD11，BIGM11，BIGS11，BIGTD1，BIGT1D，BIGT11，BIGU11．というサブルーチンへの呼び出し命令が生成される。したがってこれらのサブルーチンの本体を必要な精度を与えるようコーディングすれば良い。プログラム(B)，はそのうちの一つ，タイプBIGをもつ変数どおしの除算BIGD11のコーディング例である。ここではタイプBIGの変数を4ワードにとり，下図のように指数部，仮数部を割りつけた。



したがって指数は-2^{35}～$2^{35}-1$まで表現でき，従来の浮動小数点演算の場合（指数-2^{8}～$2^{8}-1$）に比べて表現しうる数のスケールが大幅に拡大する。

この数値の内部表現はサブルーチン COMPF からはBIGタイプ変数が4ワードであること以外は全く見えない存在である。したがって要求される計算の精度に応じてBIGOOOのサブルーチンのコーディングをしなおせば，主プログラムのアルゴリズムの変更等の面倒な問題から，プログラマーは解放される。上記引用文献には，このような大指数型数値演算に対するタイプ文の応用の可能性の指摘に加えて，倍精度複素数の演算の例が説明されている。

プログラム(A)

```
      SUBROUTINE COMPF
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      TYPE BIG*4,TMIN,FF,C,C2,W,BIGABS,B,CONST,AU, EP2,AN
      TYPE BIG*4,WW ,WJ,Z          ,BIGONE
      COMMON IB,IB1,W,P(65),F(65),TM,EPS
      DIMENSION C(65),W(65),C2(65),TMIN(65)
      DIMENSION FF(65)
      Z=W
      BIGONE=1.0D0
        :
   26 C(I+1)=((Z+WJ+AN)*C(I)-WJ*C(I-1))/Z
   25 AU=CONST/((B+AN)*C(IB1)-B*C(IB))
      DO 30 J=1,IB1
      IF(TMIN(J).LT.EP2) GO TO 35
      W(J)=AU*C(J)*(-Z)**W(J)/C2(J)
      CALL BIGABS(WW,W(J))
      IF(WW.LT.TMIN(J)) CALL BIGABS(TMIN(J),WW)
      IF(TMIN(J).LT.EP2) IFN=IFN+1
      FF(J)=FF(J)+W(J)
      GO TO 30
   35 IF(IFN.EW.IB1) GO TO 60
        :
      RETURN
      END
```

プログラム(B)

```
      SUBROUTINE BIGD11(XF0,XF1,XF2)
C     X0=X1/X2
      TYPE BIG*4,X0,X1,X2,XF0,XF1,XF2
      DOUBLE PRECISION TIS0,TIS1,TIS2
      DIMENSION IBIG0(4),IBIG1(4),IBIG2(4)
      EQUIVALENCE (X0,TIS0),(X1,TIS1),(X2,TIS2)
      EQUIVALENCE (IBIG0(1),X0),(IBIG1(1),X1),(IBIG2(1),X2)
      EQUIVALENCE (IBIG0(4),N0),(IBIG1(4),N1),(IBIG2(4),N2)
      X1=XF1
      X2=XF2
      N0=N1-N2
      TIS0=TIS1/TIS2
      CALL NORMAL (X0)
      XF0=X0
      RETURN
      END
```