寄　　書

数式向き計算機についての 二，三 の注意*

野崎昭弘**

以下に述べるのは，数式処理（compile）の簡易化に関する，筆者の思いつきの紹介である***.

数式処理の手順はすでに明らかにされている（例えば〔1〕）．それらを回路化して，計算段階（run time）に処理を実行させることも，繁雑ではあろうが，できないことはない．しかし，そうまでして飜訳時間を短縮しても，計算時間が長くなるのでは何にもならない（ループがある場合に注意）．そこで，問題を少し簡単にして，括弧のない数式の処理に限って，その範囲で何かうまい方法がないか，考えてみた．いいかえれば，括弧の前後の処理はコンパイラに任せることにして，さしあたり括弧のない数式——といっても，優先順位の規則によって省かれた，‘隠された括弧’は許すのである——を，そのまま機械語として持つ計算機はないか，ということである．まず一般的な事例から述べていきたい．

演算は優先順位に関して n 階級にわかれているとしよう．それらを，優先順位の低い順に，$\Delta_1, \cdots\cdots, \Delta_n$ とかく．つまり $i<j$ のとき，

$$x\Delta_i y\Delta_j z = x\Delta_i(y\Delta_j z)$$
$$x\Delta_j y\Delta_i z = (x\Delta_j y)\Delta_i z$$

実際には一つの階級に二つ以上の演算があってもよいが，記号的には同じ Δ_i で間にあわせよう．我々がここで考える数式は，次のようなものである．

$$x_0\Delta x_1\Delta \cdots\cdots \Delta x_N$$

さて，一つの累算器Aと，$n-1$ 個のバッファ・レジスタ $C_1, \cdots, C_{n-1}$ をもつ計算機を考えることにしよう．各レジスタ C_i は，それが情報をもっているか否か（1または0）を示す指標 T_i 各1ビットをもっている．命令には2種あって，

（1）$<x_0>$；$T_1 = \cdots\cdots = T_{n-1} = 0$ とし，Aに x_0 をかきこむ（clear add）．

（以下，C_0=A，$T_0 \equiv 1$ とする）

（2）$<\Delta x>$；

1）$\Delta = \Delta_n$ のときは，演算 Δ は直ちに実行される．すなわち，$T_h \neq 0$ なるような最大の h（以下記号 h はこの意味で使う）を求め，次の処理を行なう．

$$C_h\Delta x \rightarrow C_h$$

2）$\Delta = \Delta_i$, $i<n$ なら，次のように働く．

2−1）$h<i$ か否か？

YES らな 2-3），No なら 2-2）に進む．

2-2）h の次に大きい，$T_h' \neq 0$ なる h' を求める．

$$C_h\Delta_h C_{h'} \rightarrow C_{h'},\quad 0 \rightarrow T_h$$

を実行する．2-1）にかえる（h の値は変っていることに注意）．

2-3）$x \rightarrow C_i$, $1 \rightarrow T_i$（動作完了）

なお Δ_i と同じ階級の演算が他にもあれば，それらを区別する指標 I_i 何ビットかを C_i に附属させておかなければならない．そのようなとき，2-2）において I_h が参照されるのは当然である．

このように計算をすすめると，最終結果はいくつかの Δ_i を実行し残した形で，A(=C_0)，$C_1, \cdots, C_{n-1}$ にまたがって残る．そこで命令 $<=y>$ を設け，最終結果をAおよび y におくようにするとよい．

$<=y>$ の機能は $<\Delta_1 y>$ に準ずる．ただ 2-3）だけを次のように変更すればよい．

2-3)′　Aの内容を y にかきこむ（動作終了）

このような命令系によれば，飜訳段階での優先順位の比較は全くいらなくなる．我々の計算機は，〔4〕に述べた意味で

$$x_0\Delta x_1\Delta \cdots\cdots \Delta x_N$$

を機械語としてもっている．

Aおよび $C_1, \cdots, C_{n-1}$ を高速メモリで作っておけば，時間のむだはごく僅かである．T_h のテストが余分な手間であるが，中間結果の待避を全く必要としないので，その面からの時間の短縮も期待できる．ただし多数のレジスタを要し，コストが増すことは避けられない．

---

* Some Remarks on the Formula Oriented Computers, by Akihiro Nozaki (Tokyo Vniuersity)

** 東京大学教養学部

*** 旧論文〔3〕の一部である．

$n=2$ のばあい，特に演算が加減乗除に限られるときは，いろいろと特殊な事情があって，装置の簡単化が計れる*．Cレジスタは一つですむし，あとに示すように T, I もいらなくなるので，大いに有望かと思う．

この場合の我々の計算機は，レジスタ A，C をもち，次のような命令の備えられているものである．

<00>；A=C=0 とする．
<+x>；A+C→A，しかる後 x→C
<−x>；A+C→A，しかる後 −x→C
<×x>；C×x→C
</x>；C/x→C
<=y>；A+C→C および y，しかる後に 0→A

試みにこの方法で，

$a\times x-b\times y+c\times z=w$　　（答を w におけ）

という数式を処理させてみよう．それは下表に示すとおりになる．見られるとおり，この計算機は，括弧のない数式をそのまま機械語として扱い得る．

命令	命令実行後のAの内容	命令実行後のCの内容
<00>	0	0
<+a>	0	a
<×x>	0	$a\times x$
<−b>	$a\times x$	$-b$
<×y>	$a\times x$	$(-b)\times y$
<+c>	$a\times x+(-b)\times y$	c
<×z>	$a\times x+(-b)\times y$	$c\times z$
<=w>	0	$a\times x+(-b)\times y+c\times z$

終了後　$w=C=ax-by+cz$

計算時間を延長させるおそれは，T や I を参照する必要がないので，全くない．かえって加減算などは，記憶装置から被演算数をとりだす動作と，A+C の計算とが全く独立に行なわれるので，それだけ所要時間が短縮する．また加減乗除のみを対象にする限り，中間結果の待避の必要は全くない（$\sum a_i x_i$ という形の計算は，極めて能率よく行なわれる）．A および C を高速メモリで作ることも無理のない注文であろう．

要　約　この方法の特徴は，累算器の他に1個（または複数個）の（高速）バッファ・レジスタを設けて，優先順位に関する階級の差を，run time に認識・処理しようというものである．そのようにして，括弧のない数式を機械語としてもつ計算機の設計方式を示すことができる．したがってコンパイルの問題は，括弧のない数式に関する限り，事実上なくなる．

この方式は，命令系に関してはふつうの単一アドレス方式をとっている．そのためこれまでのプログラム技術はすべて生かされ，現在採用されている多くの命令系と compatible である．多項式計算や，多重精度算等々のための，特殊な複合命令とも同じ体系の中に含めて共用できる点は，便利といえるであろう．

補　説

筆者は電気通信研究所の電子計算機 M 1-B において，上記 <+x>，<−x>，<×x>，</x> に相当する擬似命令を有する浮動小数点演算ルーチンを作成した（〔5〕）．これには既に記された命令の他，プログラムを簡単にするための次のような命令を備えている．

<0+x>；<00> と <+x> の複合命令
<0−x>；<00> と <−x> の複合命令
<(>；A, C の内容を，穴蔵式記憶装置 S に待避させる．
<)>；A+C→(0 番地) を実行し，次に S から A, C の内容を復旧させる．
$\dot{<+x>}$；C+x→C
$\dot{<-x>}$；C−x→C

命令 <(>，<)> は括弧の処理のために設けたものである．

……Δ(……)……

のような数式は，次のようにプログラム化される．

……<(>……<)><Δ0>……

（0は‘0番地’の意味である）

命令 $\dot{<+x>}\dot{<-x>}$ は，優先順位が乗除算に等しい加減算（それらを $\dot{+}$，$\dot{-}$ とかく）を使いたいとき，利用される．たとえば，

$x^3-x^2+x-1=((x-1)\times x+1)\times x-1$
$=\dot{x}-1\times\dot{x}+1\times x-1$

これは次のようにプログラム化される．

<0+x>$\dot{<-(1)>}$<×x>$\dot{<+(1)>}$
<×x>$\dot{<-(1)>}$

ただし（1）は定数1の所在番地をあらわす．

このような命令系を試用したところ，簡単なアセンブラによっても，数式の計算のプログラミングが比較的容易に行ない得ることがわかった．

なお，論文〔1〕は何分古いもので，最近の情報は何も盛られていない．また括弧の前後の処理を run time に行なうことについては，私なりの疑義もあり，深くは考えていない．金山氏のような立場から見れば，極めて狭い視野に立つ考察である．ここに述べられていない，いろいろな着想もあることと思うが，この誌上を通してでも御教示いただければ，まことに幸いである．

〔注〕‘事情’の詳細については，〔3〕Ⅲ．§3 および注 4, 5 (pp. 39〜40) 参照．

参考文献

1) Sequencial Formula Translation; K. Samelson, F.L. Bauer, Com of the ACM (1960-2)
2) ALGOL ORIENTED COMPUTER; 金山裕，情報処理学会全国大会講演予稿集 (1963)
3) 自動プログラミング I; 野崎昭弘，電気通信研究所成果報告第 1741 号 (1962)
4) 数式向計算機; 野崎昭弘，情報処理，Vol. 5 No. 2 (1964)
5) M 1 ライブラリ，補充サブルーチン集 (2); 野崎昭弘，電気通信研究所経過資料

（昭和 39 年 2 月 10 日受付）