Automatic Coding について (IV)

—— FORTRAN Compiler (4) ——

穂 鷹 良 介

総 目 次

1969年4月19日受領

5. 5. 入出力命令

入出力命令の compile を考えるときに一番問題となるのは入出力並びと書式制御の compile 方法である。その他の面でも実際の compiler 作成に際しては，入出力機器の多様性の故に統一的な取り扱いが難かしく，debug に労力を要する所が多々あるが，machine dependent な面がすくなくないので詳しい検討は上記二つの問題及びそれに関連する部分だけに限定する。従って具体的には書式 (FORMAT) つきの READ 文又は WRITE 文で，しかも入出力並びのあるものの compile 方法を念頭におく。

compile の手続きは大きくわけて次の三つの段階にわかれる。即ち初期化を行う段階，入出力並びを記憶する段階及び，その記憶した入出力並びに対応して object program を作り出す段階である。これらの各段階の説明に先立って実行時に用意されるサブルーチンから述べる。

5. 5. 1. 入出力解釈ルーチン

入出力命令がどのように書かれたとしても，それらの命令は別に用意された基本的な一連の操作の組合せとして表現出来るならば，入出力命令の object program はこれらの基本操作をうまく組合せていく main program の形をしているだけでよい。我々の compile の方法もこのような方法をとることにし，この別に用意された基本的な一連の操作を入出力解釈ルーチンとしてまとめたもので 1. でのべた各モード処理プログラムの一つをなす。これには機能的にみて二つの部分がある。

(1) 初期化部分

この部分は main program から渡された情報により，何の入出力命令かを判別し必要な初期化を行うと共に，関係するプログラムをコア内部に準備し，入力の場合にはデータの格納番地，出力ならばデータの所在番地を受取りに再び main program にコン

トロールを渡すルーチンである。その際に次の入出力処理を行う実行ルーチンの先頭番地を EBAC（2バイトの作業エリア）に指定しておく。

(2) 本体部分

この部分は初期化部分或いは本体部分で戻りの位置が指定されているときに，main program の方からコントロールを渡すことによって動き始める。その際に次の処理を行う main program の先頭番地を EBAC に再び指定しておくものとする。ここでは main program から情報を貰い，入出力を行う。main program から渡される情報は入出力並びに関するもので，入出力命令は main program と入出力実行ルーチンとの間に EBAC を仲介として何回かコントロールが往復することによってなされる。つまり，実行ルーチンは入出力並びを単位要素ごとに処理し，main program は次々と入出力並びに従って入出力並び要素の情報を作り出すのが夫々の役目である。

5. 5. 2. 入出力命令の初期化処理

以下では実行時に 5. 5. 1. でのべたような実行ルーチンが用意されるものとして，入出力並びの compile 方法（5. 5. 1. で述べた main program の作り方）を述べる。

この段階では入出力命令を分類して，‘最初のコントロールを入出力実行ルーチンの初期化部分に渡す’ object program を作り出す。

5. 5. 3. 入出力並び処理 (stack)

ここでは並びを標準形に整えてから，table（これを list table と呼ぼう）に stack する。カッコ，繰り返し指定の取り扱いには別に parenthesis table, loop table を設けて，次のように情報を蓄える。

例. ((A(I, J), I=1, 3), J=1, 5, 2)

list table に書かれる情報

$p_1\rightarrow$ (の syllable, ad_1

$p_2\rightarrow$ (　　〃　, ad_2

　　A　　〃

$p_3\rightarrow$ (　　〃　, ad_3

　　I　　〃

J の syllable

$ad_3 \rightarrow$)　　〃　, ϕ

$ad_2 \rightarrow$)　　〃　, ad_4

$ad_1 \rightarrow$)　　〃　, ad_5

loop table に書かれる情報

$ad_4 \rightarrow$ Ad(I), Ad(+1), Ad(+3), Ad(+1)

$ad_5 \rightarrow$ Ad(J), Ad(+1), Ad(+5), Ad(+2)

parenthesis table に書かれる情報

p_1

p_2

p_3

一般論はやめて上のごく簡単な例についてどのように三つの table を作成していくかを説明する。入出力並びも PHASE II で syllable の分解がなされているから，list table にはその syllable を順に書いていく。'(' が来たときには，対応する右のカッコ ')' が出るまで，その list table 上の位置 (p_1, p_2, etc.) を parenthesis table に順に書いていく。閉じカッコ ')' が現われたときには，parenthesis table をみて最後に登録された list table 上の位置の示す list table の element に今現われた閉じカッコ ')' の list table 上の格納位置 (ad_3, ad_2, etc.) を追加する。

つまり，左カッコの syllable 後に，対応する右カッコの list table 上での位置がさかのぼって書かれる。

繰り返し指定が現われたときには，必らず右カッコが現われるから，この右カッコに対して上と同様の処理を行い，更に I=1, 3 に対応する情報として，DO 文の扱いと同様の情報 (Ad(I), Ad(+1), Ad(+3), Ad(+1)　夫々 I, +1, +3, +1 に割りつけられた address) の組を別の loop table に格納し，その address (ad_4, ad_5) を list stack の閉じカッコ ')' の syllable の後につけ加えておく。繰り返し指定に関係のない閉じカッコ ')' の場合この address 部には ϕ の記号を入れておく。これらの右カッコの処理が終えるたびに一組のカッコの処理が終ったとして parenthesis table の内容は最後のものを一つずつ消していくものとする。

並びの中に配列名が出て来た場合には，単に基準 address 又は引数の address とあとは配列の全体の大きさの情報を list table に書き残しておくだけでよい。なお以上の処理で後の処理に不要な ，の syllable はよみ捨てる。

5. 5. 4. 入出力並び処理 (compilation)

入出力並びの終りに到達した所で，今迄 list table と loop table に作成した情報を用いて，並びの要素を次々にとりだす main program に当る部分の compilation を行う。list table を最初からよんでいって，左カッコ '(' が現われたときには対応の右カッコをしらべ，それが繰返し指定に関係のあるときには，DO 文の時と同様に loop 形成の初期値設定の object program を出力し，同時にこの位置を Return address table に記憶する。loop 形成に関係のない左カッコについては何もしないで，次の list table の処理に進む。

右カッコ ')' が現われた時には DO の端末文の処理と同様の loop 形成命令を作成すると共に Return address table の内容は last-in の element を一つ削除する。

配列名のみの並びが出たときには，次元に関係なく，その配列の最初の元の address と最後の元の address とを知り，あたかも 1 次元のデータをとりだすような object program を出力すれば良い。

他の list table の要素は一つの並びとしてその情報を実行ルーテンに向って送り出す object program を出力する。

以上で入出力並びの処理方法を DO-implied list の処理方法も一緒にして述べた訳であるが，小さい computer では 5. 5. 2., 5. 5. 3., 5. 5. 4. の処理は互いに overlay させることが可能である。list table, loop table の内容は 5. 5. 3. から 5. 5. 4. に進むときこわされてはならないが，parenthesis table の内容は正常な statement の処理が終ったときは空になるから，5. 5. 4. では何に使用しても良い。

5. 6. 数　式

JIS FORTRAN 3000 程度の数式本体の翻訳には以上でのべように syllable 分解の作業をあらかじめ行っておけば，object program の何等の economization もしくは optimization を行わない時には 128 バイト位の table を含めて大体 3600 バイトの記

憶容量で overlay を行うことなく実行出来る。

数式翻訳は次の4つの stage にわかれる。

stage 1. 初期化

数式は閉じたサブルーチンとして使われる場合もあるので，この段階でその区別をする他，各種の counter 類の初期化を行う。

stage 2. syllable read

syllable をよみ，それが delimiter でなければ operand として table に stack し，delimiter であるならばそれぞれの処理ルーチンにコントロールを渡す。

stage 3. 判定部分

処理を渡された各 delimiter の処理ルーチンは last-in の delimiter を取り出し，二つの delimiter の組に対して適当な行先にコントロールを渡す。そのコントロールの行先で，compilation が部分的にでも可能か否かをしらべ，可能ならば stage 4 に進む。compilation が完了したときには，数式翻訳が閉じたサブルーチンとして使われたか否かを調べ正しくコントロールを戻す。まだ完了していないときには delimiter を一旦 table に stack して stage 2 に戻る。

stage 4. Code generation

ここでは前に stack されていた delimiter を operator として object program の code generation を行う。等号の code generation が終ったときにはそのまま数式翻訳は完了となるが，さもなければ，今現われた delimiter をそのままの状態に保ったまま再び stage 2 に戻る。

stage 2, 3 は単純には jump table を使用して機械的に行えば楽であろう。stage 4 では accumulator の待避，非待避の別，使用された場合，結果の数値の整数型或いは実数型の別，又，演算が可換，非可換の別の状態によって若干 accumulator 使用の optimization を考えると，table に stack されている operand の扱いや accumulator の内容の待避に関係するサブルーチンが数種類必要となり，又それらの関係を表示する indicator を設ける必要がある。

5. 7. JOB END 処理

一つの FORTRAN プログラムの compilation が終ったときになさねばならぬ事柄

を列挙する。

(1) JOB 単位の table の listing を行う。これには定数表，サブプログラム表がある。

(2) object program の初期化ルーチンが必要とするデータを同じ object program 中に書きこむ。(定数表，使用サブプログラムで execution 時 core 常駐のもの等は実行に先立ってメモリに load しておかねばならぬが，これらの情報は JOB の終りでないと判らない。)

(3) 定数表，組込み函数等を object program に追加する。

(4) 使用サブプログラム等で未定義のものがないかをしらべる。

(5) object program の listing と複製を行いコントロールを次の JOB (compile and go の時にはすでに出来上った object program) へ渡す。

6. 実行ルーチン

実行ルーチンとここで呼んでいるものは，object program の実行に際してその助けをなすもので，compile されて出来上る main program と異って，どの object program にも必らずくっつけられる object program の一部である。

その機能はすでに 1. 及び 5.5.1. で述べられているのでここでは，それ以外のことについて説明を追加する。

6.1. 共通ルーチン

(1) 初期化部分

object program の真先にコントロールが渡される部分で，ここでは以下のプログラムの実行に先立って定数表，使用組込み函数，メモリ常駐サブプログラム等をコアに load し，コントロールを object program の先頭に渡す。

(2) コントロール関係

Branch 命令を行うもの，サブプログラム間の link を行うもの，エラーの際の処理を行うもの等がある。

コアは overlay されるので，これらのコントロールは overlay area とは無関係の所で行われ，サブプログラムの calling は戻り先を table に stack していく方式をと

る。

(3) 添字の計算，整数演算

(2) 10進2進変換ルーチン

6．2．組み込み関数

筆者の経験のある初等関数の coding についてのみのべる。方法は SQRT(X) を除いて皆多項式近似である。単精度演算でやった結果はいずれもうまくなく倍精度演算を使ってはじめて満足出来る結果を得た。

(1) $\tan^{-1}x$　変数 x を $0\leqslant x\leqslant 1$ に変換して近似する。$x<0$ のときには $\tan^{-1}x=-\tan^{-1}(-x)$ により，$x>1$ のときには $\tan^{-1}x=\frac{\pi}{4}+\tan^{-1}\frac{x-1}{x+1}$ により計算を実行する。近似公式は例えば［5］p. 126　9° 参照。

(2) $\sin x$, $\cos x$　sin 関数の周期性を利用して一旦変数を $-\frac{\pi}{2}\leq x\leq\frac{\pi}{2}$ に変換した後多項式近似を行う。$\cos x$ は

$$\cos x=\sin\left(\frac{\pi}{2}-x\right)$$

により計算する。

$x+\frac{1}{2}\pi\geqq 0$　のときには　$n=\mathrm{IFIX}((x+\frac{1}{2}\pi)/\pi)$

$x+\frac{1}{2}\pi<0$　〃　$n=\mathrm{IFIX}((x+\frac{1}{2}\pi)/\pi)-1$

として

$$\sin x((-1)^n x+(-1)^{n+1}n\pi)$$

を $\sin x$ の値として計算する。近似公式は例えば［5］p. 102 B) 参照。

(3) $\sqrt{x}$　Newton の方法による。$x=y\times 10^n$　$0<y<1$, n は整数となる y, n を決定し，n が偶数のとき iteration の出発値を1として $\sqrt{y}$ を求め，答の指数を $\frac{n}{2}$ ふやす。

n が奇数ならば iteration の出発値を10として $\sqrt{10y}$ を求め答の指数を $\frac{n-1}{2}$ ふやす。

7．Compier 作成作業について

FORTRAN Compiler だけでなく，一般に compiler の作成に際して気をつけた方

が良いと思われることを思いつくままに列挙してみよう。

(1) 作業日程，作業順序について。

一つの compiler の作成に必要な工程は一定ではなく，完成迄に時間がすくないと延べ動員数にしてより多くかかる。その理由は完成を急ぐときには多人数で programming を並行させるため，一部分だけ切り離してプログラムを check するときには前後のつなぎの部分だけを余計に作らなくてはならぬからである。つまり test のために作る test program に余力労力をとられる結果となるのである。従って予定を立てるときにはこの test program を作る手間が出来るだけ省けるように仕事を計画すべきである。

(2) program の経歴について。

プログラムは完全に完成する迄に何度もその内容が改変される。従って system program の経歴をはっきりしておかないと，どの時点の system program に対してどの test が通らなかったかということが明確でなくなる。debug 時には常に結果の一部に第何版の system program によるものかを明示した情報をつけるべきである。

(3) doucmentation について。

coding は特にそれが機械語でなされたときには他人には仲々その意味がわかりにくい。従って assembler 語で coding するときには 1 step ごとにその意味を comment として書いておく必要がある。これは他人に対してでなく自分に対してもその必要がある。有益な documentation は debug の speed を増す。例えば FLOW CHART と coding sheet とが直ちに対応がつく状態が望ましい。

(4) debug の方法。

debug 用のプログラムは完成プログラムに残らないからといって，おろそかにしてはいけない。

虫が発見されたときに効果的にその虫を取り除くための debug の手段をプログラムの設計と同様に真剣に考えておくべきである。

特に同じ test を何回も最初から短かい時間で確実に繰り返すことが出来るようになっていなくてはならない。

8. JIS FORTRAN 3000 との相違点

以上のべて来た compiler の文法は以下の点で JIS FORTRAN 3000 [1] と相違し

ている。これ以外にも相違している所が多々あろうかと思うが，compiler system の設計方法とあわせて，御批判下されば幸いである。

(1) 非実行文中宣言文はすべて実行文の前になければならない。

この制限はもう一回 PASS の回数をふやせば，とることが可能である。

(2) JIS FORTRAN 3000 では英字名は最大5個の英数字より成るとしているのに対し，本稿では，最大6個の英数字より成るものとした。

(3) JIS FORTRAN 3000 の宣言文の現われ方の順番に関する制限を取る。

(4) common block を使用する場合，最初に出て来るプログラムの common block の大きさが最大でなければいけない。

この制限も，PASS を一回ふやせばとり去ることが出来よう。

(5) EQUIVALENCE ステートメントにおいてはエレメントの表現は1次元でしなければならない。

参 考 文 献

[1] 日本電子工業振興協会: 日本工業規格原案
電子計算機プログラム用言語 FORTRAN 3000　41年10月
[2] 日立製作所 HITAC 5020, 5020 E/F FORTRAN (HARP) 5020－3－005－02
[3] 森口繁一 FORTRANIV 入門 東京大学出版会 1965年
[4] 穂鷹良介 Automatic Coding についてⅢ(1)and (2) —COBOL Compiler— 商学討究第18巻1，2号
[5] Л. А. Люстерник, О. А. Червоненкис, А. Р. Янлольский, Математический Анализ.
Физико-Математической Литературы МОСКВА 1963.