Automatic Coding について (IV)

—— FORTRAN Compiler (1) ——

穂鷹良介

Compiler 言語として最も歴史が古く，また，使用頻度も多い FORTRAN については，数多くの著者がいろいろの角度から，研究成果を発表している。しかし compiler を作る具体的な方法については，計算機に依存する面が多かったり，あまりきれいごとではすまないせいもあってか，細部の点まで公表されている設計書に類するものはすくないようである。（勿論各製作者はその compiler の設計書なるものを持っているのに違いないのだが，compiler の提供を受けたユーザーが，それを要求したとしても簡単には入手出来ないもののようである。）

そのため，プログラミングの勉強をしている研究者が，新しく一つの compiler system を作る際には，プログラマー仲間で口伝えにされているようなことを土台として，各自が工夫をこらして無手勝流で始めるということになりかねない。

本稿では，そういった FORTRAN compiler の設計書に当るものを主に以下の PHASE III を中心にして述べ，新しくシステムプログラムを作ろうとしている人の参考に供したい。

FORTRAN の文法としては［1］（JIS FORTRAN 3000 と略称する）程度のものを考えるが，完全には一致してないかもしれない。気付いた相違点は箇条書にして本稿の最後で一括して述べる。計算機としては小型のものを考える。従って特に内部メモリの不足が compile 方法に与える影響については詳細に吟味する。記憶装置としては，バイト単位の variable length

の内部メモリ（大体８Ｋバイト以上）と比較的大容量の補助記憶装置（例えば DISK, DRUM）を有しているものを一応念頭におくが，compile の方法自体は，他の計算機にもあてはまることが多いであろう。

目　　次

1. Object Program

object program の設計方法にも多様な方法が考えられるが，本稿では内

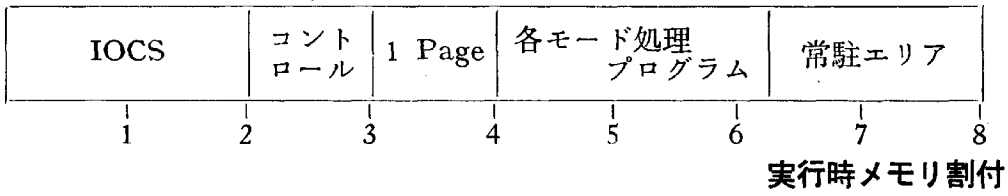
部メモリが比較的小であることを勘案して，次のようなものを考える。

(1) object program は常駐 subprogram，実行時各種コントロール部分を除いて page 式にして，1 page に入りきらないような object program は source program に何の指定を書かなくても，自動的に次の page にまたがるようにする。コントロールが複数 page にまたがるとそれだけ計算時間は長くかかることが予想されるから，programmer が考えて，1 page を効率的に使うために，指定によって，新しい page に切換えることも可能であるようにする。なお，main program より前に置かれた subprogram のみを内部メモリ常駐とする。

(2) 入出力コントロールルーチン，変数エリア，定数エリア，組込み関数，programmer の指定する常駐プログラム，object program 実行コントロールルーチンは常に内部メモリに滞在させる。

これに対して各種入出力解釈ルーチン，浮動小数点四則演算，巾乗ルーチン，log x, e^x, FLOAT(x), IFIX(x), 等の算術演算が必要とする SR（サブルーチン）類，デバッグルーチン etc. は内部メモリに一度に全部は入れないで，実行段階を演算モード，入出力モード，デバッグモードにわけ，それぞれのモードに合った処理プログラムを，そのつど内部メモリによみこむこととする。勿論計算機の内部メモリに十分余裕がある場合には，これらの考慮をする必要はなく，すべての処理プログラムを実行段階を通じて内部メモリに常駐させておけばよい。

以上の考慮から，実行時の内部メモリのメモリ割付を考えるならば次の図の如くなろう。



実行時メモリ割付

IOCS：入出力コントロール部分

コントロール：page のオーバーレイ，SR 間の link コントロール添字計算ルーチン等

1 page：object program のオーバーレイエリア

各モード処理プログラム：演算処理プログラム，入出力解釈ルーチン，デバッグルーチン等のオーバーレイエリア

常駐エリア：この部分は COMMON エリア，常駐サブプログラムエリア，定数エリア，変数エリアのために使われる。

2. Compiler 概説

2．1. Compiling 方法の決定

1. で述べたように object program の仕様が定まれば，次にこれを実現する compiler をいかに設計するかということになる。

この種の計算機の compiler を設計するにあたって考えねばならぬことがすくなくとも二つある。

その 1 は，処理プログラムの分割についてである。内部メモリがあまり大きくない場合，またはかなり大きいとしても，内部におく table 等を大きくとったため処理プログラムの入るスペースが，実際上小さくなったような場合には，必然的に一時に内部メモリに入る処理プログラムの大きさは制限されてしまうから，プログラムを機能的にまとまった単位ごとに分割する必要が出て来る。これをどの機能別にどの位の大きさに分割するかということが一つの問題で，これは次の問題とも密接にからんでくる。

その 2 は compiler が source program ないしは，順次変形された source program を何回 scan するかということである。つまり PASS の回数である。当然のことながら，各 PASS ごとにどのように source program を変形していくかということも第 1 の問題とからみ合わせてここで決定されねばならない問題である。

PASS の回数を極端にすくなくして，例えば 1 回とする（1 PASS 方式）

と，source program をよんだ時点で，何から何までの操作をやらねばならず，当然のことながら，一回目の PASS では compiler にとって未知の情報は，あとで適当に埋めこむことが出来るような状態（いわゆる芋づる操作の出来る状態）にして，一旦 object program を出力しなくてはならない。

また，この方式では，source statement の現われ方によって，そのつど，対応する処理プログラムをオーバーレイさせて内部メモリによみこまなくてはならないので，補助記憶装置の速度が遅いと compile 速度が遅くなる可能性がある。

これに対して PASS の回数をふやすことは PASS ごとに変形していく中間結果の仕様を入念に設計しなくてはならず，処理プログラムのオーバーレイは 1 PASS 方式に比べて，すくなくすんでも 1 回の PASS に要する処理時間がかさむから，PASS の回数もあまり多く出来ない。

以上のような観点から，これらの中間をとって，次のような compile 方法について考える。

PASS 回数は 3 回とする。

第 1 回の PASS では source program を入力装置より読みとり，statement の種類を判別して，分類番号をつけて補助記憶装置に格納する。同時に指定に基いて，source program の listing（製表）も行なう。

第 2 回の PASS では，この第 1 回の PASS 後の分類番号つきの source program をよみとり，各 statement をシラブルに分解して，再び補助記憶装置に格納する。この段階では変数，定数の割付を完了し，文関数，subprogram に関するメモリ割付も行なう。従って，各変数の name table, subprogram table, 定数 table が用意され，使用される。

第 3 回の PASS では，第 2 回の PASS 後のシラブル分解の結果と，subprogram 表を使い object program の code generation を行なう。この段階では更にステートメントナンバーの処理を行なうため statement number table が用意され，使用される。

これらの第1, 2, 3回の PASS とそれに伴なう処理をそれぞれ PHASE I, II, III とよぶことにしよう。

compile はこれらの PHASE を各プログラムごとにくりかえして行なうことによってなされる。

2. 2. 各種 table

以上のような compiling 方法を採用するときに用意しなくてはならない table について述べる。これらの中味は当然 **3.** 以下の各 PHASE の検討の結果きめられるものではあるが，各 PHASE を通じて使われる table もあるので，一括してここでその概要を述べる。

2. 2. 1. Name Table

name table は次の element より成る。

name	D	P	S	ad	L	LM	LMN

name : 6バイト　変教名（配列名も含む）の名前

D　　1バイト　この 1 element の長さ

P　：$\frac{1}{2}$バイト　各 bit に次の意味をもたす

2^3, 2^2 bit : 変数の次元を set する

2^1　bit : 引数のとき1それ以外のとき0

2^0　bit : 使用せず

S　：$\frac{1}{2}$バイト　各ビットに次の意味をもたす

2^3 ビット : name のメモリ割付が完了したとき1

2^2 ビット : name が用いられたとき1

2^1 ビット : name が EQUIVALENCE 宣言で用いられた時1

2^0 ビット : name が COMMON 宣言で用いられた時1

上の条件が満たされないときには，各ビットは0と set されている。

ad：2バイト　この name に対する基準番地を set する。

基準番地は次のように定められる。

単純変教または FUNCTION name のとき ad＝単純に割付けられた番地。

1次元の配列名のとき　$ad=l(a_1)-1$

2次元の配列名のとき　$ad=l(a_{11})-L-1$

3次元の配列名のとき　$ad=l(a_{111})-LM-L-1$

但し a_1 は1次元配列の第1 element, a_{11} a_{111} についても同様とする。$l(a_1)$ は a_1 に割付けられた番地を示す。$l(a_{11})$, $l(a_{111})$ についても同様とする。

L　：2バイト　第1添字の寸法の大きさ

LM　：2バイト　第1添字の寸法の大きさと第2添字のそれとの積

LMN：2バイト　第1，2，3添字の寸法の大きさの積

なおこの表は D part で示す通り，1 element の大きさが可変長で，例えば，name がはじめから単純変数ということが判っているようなときには，L, LM, LMN part がない形で，name table を構成する。

name table はプログラムごとに独立しているものであるから，各プログラムごとに PHASE II のはじめに新規に作られる。その結果は PHASE III の終りまで保持されるが，それは listing に使用するだけで，PHASE III でこの table 内容を直接参照することはない。name table の情報で，PHASE III の compile のために必要な部分はすべて PHASE II のシラブル分解の際に盛りこまれる。

2. 2. 2. Statement Number Table

table の element は次の bit pattern を持つ。

n	q	p	x / ad

n：3バイト　ステートメントナンバを set する。

q：$\frac{1}{2}$バイト　各 bit に次の意味を持たす。

2^3 bit：入出力命令に関する statement number の時1。

2^2 bit：この statement number の使用を禁止するとき1，この bit を禁止 bit という。

2^1, 2^0 bit：使用せず。

p：$1\frac{1}{2}$バイト　compile 中に2通りの使い方がなされる。

(A) statement number の位置が確定したとき，その確定した page 数を set する。

(B) statement number の位置が未確定のまま使用された場合には，この statement number を使用した最後の statement の位置を object program の page 数でかく。（この部分は ad 部分とあわせて芋づる情報をなす。）はじめて使用されたときは0の値が入る。

ad：2バイト　compile 中に2通りの使い方がなされる。

(A) statement number の位置が確定したとき，その確定した address を set する。

(B) statement number の位置が未確定のまま使用されたときには，はじめて使用された時には0の値，さもなければ最後にこの statement number を使用した statement の object program の address を set する。

x：1 bit ad part の左端の 1 bit を使用する。

statemant number の位置が確定したとき0，未確定のときは1。

statement number table も name table 同様プログラムごとに独立しているものであるから，各プログラムごとに PHASE III のはじめに新規に作られる。その使い方等については更に **5. 4.** で述べる。

2. 2. 3. Do Table

Do table の element は次の bit pattern を持つ。

n	r	p	ad	loc	I	m_2	m_3

これは FORTRAN の source statement

DO n I=M_1, M_2, M_3

（但し M_3 がないときは M_3=1 があったものとみなす）に対応して作られるものである。

n：3バイト　statement number

r：1バイト　DO の nest の深さ

p：2バイト　DO の terminal statem nt 終了後 loopを形成するためにコントロールを戻す戻り先の object program の page 数。

ad：2バイト　同上 address

loc：2バイト　この DO の処理を行なう時点での statement number table の location counter

I：2バイト　DO の制御変数 I に割付けられた address, 引数のときには左端の 1 bit が 1 に set される。

m_2：2バイト　DO の終値 parameter に割付けられた address, 引数のときには左端の 1 bit が 1 に set される。

m：2バイト　DO の増分 parameter に割付けられた address, 引数のときには左端の 1 bit が 1 に set される。

DO table もプログラムごとに新規に作られる。使い方等は **5. 4.** コントロールステートメントの項で述べる。

2. 2. 4. Subprogram Table

subprogram table の 1 element は次の bit pattern を持つ。

name	n_p	s_p	ad_3	p	ad_1

name: 6バイト　subprogram の名前を set する。

n_p　: 1バイト　subprogram の引数の個数。

s_p　: 1バイト　各 bit に次の意味をもたす。

2^7: subprogram が組込み関数 COS の時 1

2^6: subprogram が PHASE II 段階で定義された時 1

2^5: subroutine の時 1

2^4: ステートメント関数の時 1

2^3: 組込み関数の時 1

2^2: 外部手続で内部メモリに常駐する時 1

2^1: 使用された時 1

2^0: table の内容がすべて確定した時 1

上の条件が成立していない時，各 bit は0に set されている。

ad_3: 2バイト　この subprogram に割付けられた address

p: 2バイト　compile 中に2通りの使い方がなされる。

(A)　この subprogram に関するすべての情報が確定したとき。

　　この subprogram の object program のページ数。

(B)　この subprogram の情報が未確定のまま使用されたとき。

この subprogram がはじめて使用された時には0の値，さもなければ最後にこの subprogram を使用した statement の object program のページ数を set する。

ad: 2バイト　compile 中に2通りの使用がなされる。p と同様であるが page の代りに address が set される。

subprogram table は 1 JOB を通じてプログラムごとに clear されることなく使われる。なお，**4. 3.** 参照のこと。

2. 2. 5. 定数 Table

定数 table の element は定数 1 個で，1 語（FORTRAN で扱う数値の単位，6 バイト位が適当である。）より成る。この table は compile 時（PHASE II）に作られたものが，実行時にそのまま用いられる。

2. 3. 各種カウンタ，Indicator 類

compile を行なうには各種のカウンタを設ける必要があるが，カウンタは indicator と並んで，もっとも system の bug になりやすいものであるから，その取扱いは慎重を要する。

variable length の機械で，（しかも index register がないような場合は特に）ある位置を表示するカウンタは，その取扱い方法を次のように統一しておくと間違いを防止する上に役立つ。

＜カウンタの原則＞ low location から high location に向ってカウンタがふえるような使い方をするときには，使い終った状態では常にカウンタの値が，次の位置を指しているものとする。

逆に high location から low location に向ってカウンタの内容がへるような使い方をする時には，カウンタの内容は使い終った状態で，処理し終った情報の一番低い location の位置を指しているものとする。

JOB を通じて必要と思われるカウンタ，indicator は次の通りである。

AERR : source program にエラーがあったとき 2^0 bit を 1 に set する。この bit が ON のときは execution に入らない。

AWC 2 : object program を格納しはじめる補助記憶装置の位置。この内容は JOB が終った所で，補助記憶装置内の object program をこわしたくない時には，適当に raise される。

AVIX：変数を割付けるために使う。

AMIX：COMMON エリア，常駐 subprogram の割付に用いる。

AREG：object program の main program 第1ページのページ数。

SYPC：compile 中のプログラムが内部メモリに常駐すべきものか否かを示す indicator。

ANAM：name table のカウンタ。

APID：object program のページ数のカウンタ。

ACON：定数 table のカウンタ。

ASUB：subprogram table のカウンタ。

3. PHASE I (注)

PHASE I では source program を入力し，指定に応じて listing を行ない，statement の種類を判別する。source program には statement の種類を現わす番号をつけて，一旦補助記憶装置に出力する。

なお，この PHASE では JOB の初期化をあらかじめ行なわなければならない。object program の頭には，実行段階で使用する各種の SR 類，初期化ルーチンをつけるものとし，それらを object code generation に先立ってこの PHASE で行なう。

この PHASE の主目的は FORTRAN statement の分類であるが，この作業は FORTRAN に COBOL でいうところの reserved words の約束がないためにいろいろと面倒な手続きをふむ必要がある。

statement を分類する方法を考える前に分類の際救わねばならぬ特殊ケースを若干あげてみる。

例 1　GO TO 10=1

は数式である。

(注) FORTRAN statement の分類に関しては北大工学部助教授栃内香次氏に貴重な経験をきかせていただいた。

例 2　GO TO (10, 20)=I

は GO TO という name がすでに配列の中にみつかれば，添字の計算をする数式である。

例 3　READ (5, 10)=I

これも READ という name が配列の中にあれば添字の計算をする数式である。

例 4　DO 30 I=1, 15

は普通の DO statement であるが

例 5　DO 30 I=1.15

は数式であって，DO30I という変数に 1.15 という値を代入する操作を表わす。FORTRAN では FORMAT statement の Hollerith 表示以外においてはブランクを無視しなくてはならないのでこんな面倒が起きる。

例 6　FORMAT (I2, H8)=(I2+H8)

はステートメント関数であるから PHASE I では数式に分類すべきであるが，

例 7　FORMAT (I2, 8H)=(I2+H8)

は FORMAT statement として分類しなくてはならない。

これらの例の考察からも判る通り，分類でまず第 1 に考えねばならぬことは数式の分離である。従って statement の分類方法としては大体次の手続きを行なうのがよかろう。

(1)　source program の 7 カラムよりブランクはよみすてて一字ずつ読み

とる。まず＝か否かを判定し，＝ならば DO の判定(2)に行く。（ならば(3)へ行く。

変数名は6文字をこえることがないから，この操作が6回を超えたときには，6文字を超える FORTRAN の key word であるとみなしてブランクをとってつめた key word を set して分類番号決定(6)に行く。

6文字をこえぬ内に 1 statement の終りになったときにはこれも key word とみなして分類番号決定(6)に行く。

(2) ＜DO の分離＞ 一字ずつブランクをよみすてて，よみとる。まずカンマ ',' か否かを判定し一致ならば DO statement とする。次に（か否かを判定し一致ならば，数式とする。いずれも現われない内に statement の終りとなれば数式とする。

なお DO 1000 I＝1, 10 という DO statement は，この段階では分離されず，分類番号決定(6)で決定される。

(3) いままでブランクをつめてよんだ6文字が FORMAT ならば (4) に行く。

一字ずつよんでいって，途中で statement の終りとなれば (6) へ行く。途中で＝が現われれば数式とみなす。

(4) Hollerith indicator を設ける。一字よみとり，これが数値ならば indicator を ON にし，さもなくば OFF の状態にする。/ ならば FORMAT statement とする。

(5) 一字ずつよみとり，カッコ '(', ')', カンマ ',', ならば (4) に行く。＝ならば数式とし，1 statement が終了したときには FORMAT とみなす。スラント '/' ならば FORMAT とし H が現われたとき Hollerith indicator

が ON であれば FORMAT とする。数字以外の文字であれば Hollerith indicator を OFF にしてこの手続き(5)を続ける。

(6) key word を table の中からさがし，分類番号を決定する。

(6)で table を search する際には

IF(A) 10, 20, 30

と

IF ACCUMULATOR OVERFLOW 10, 20

等とを混同しないように前者には（をつけたままの形で key word table の内容と比較する等の工夫が必要であろう。

なお statement 関数は以上の方法では数式に分類されるが，この識別は PHASE II で行なう。

上の分類手続き中(4)，(5)は要するに FORMAT（と始まる statement が数式か否かを判別している部分である。JIS FORTRAN 3000 [1] では FORMAT statement は次の形をしているものとしている。

FORMAT (q_1 t_1 z_1 t_2 z_2 ... t_n z_n q_2)

但し

(q_1 t_1 z_1 t_2 z_2 ... t_n z_n q_2)

は書式仕様で

q_1 と q_2 とはいくつかの斜線または空

t_1, ..., t_n はいずれも欄記述子か欄記述群

z_1, ..., z_n はいずれも欄区切りである。

この説明からみる限り

FORMAT(3X5HA)=(B)

なる statement は正しい FORMAT statement ではないように思われる。

しかるに同じような文法規則を FORMAT statement について述べてい

る［2］p. 33 において

FORMAT (1Hb, 3X5HABCDE, F7.31)

という例があげられているし，また［3］p. 83 においては『nx や n H h_1 h_2 ... h_n で記述される field のあとなど，まぎらわしくないときはコンマはなくてもよいが，書いてもよい』とある。われわれの上で述べた方法では(5)で区切り符号を検出して(4)に戻りそこで Hollerith indicator が適当に set されるのだが，上のような例題は，区切り符号がないため数式と分類され，数式の compile に入ってから，エラーとみなされることになろう。従って HARP におけるように，上の例題を FORMAT とするためには，(5)で「H または X が現われたとき Hollerith indicator が ON であれば FORMAT とする」ように変更すればよいであろう。JIS FORTRAN 3000，あるいは HARP 文法から考えて上のようなケースは文法違反であると，筆者には自信をもって断言することが出来ないが，もし文法違反であるとすれば，上のような statement が数式の書き方のエラーであると compiler にいわれても致し方ないであろう。もっともこういうケースこそ［3］でいうところのまぎらわしいときであることには違いない。

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　昭和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　FORTRAN IV 入門　東京大学出版会　1965年