

自動プログラム “EASE” について*

田 村 康 男** 木 寺 達*** 柵 木 堅****

1. 結 言

工学の一分野において各種の数値計算結果を短期間に欲しい使用者の一人として、曲りなりにも動く自動プログラムを作製しようということになった。幸い手近に使える NEAC-2203 の基礎装置の範囲で、かつ限られた keyboard の活字を用いて標題の自動プログラム “EASE” (=Efficient Autocoder for Saving Efforts) を作製した。形式には特にとらわれなかったが、FORTRAN 形式に近い。“Efficient” の名の示すとおり、数式の編集・処理については、プログラムが直接機械語で coding した場合と近くなるように留意してある。

2. 自動プログラム “EASE” によるプログラム例

自動プログラム方式の文法はあくまでも製作当事者の取り決めであって、これを 1 カ条ずつ述べるのは効果的でないので、詳細を述べる前に先ず例題により、“EASE” の表現が全体としてどんな感じのものであるかを示すことにする。

例. 平方根 (ニュートンの方法)

```
VARIABLE
FLOAT, X, 2, A, 1,
FIX,
4 FORMAT (9, 2, 2)
3 READ (1) A,
X#.1 E 20
1 X1# (A/X+X)※.5
IF (ABS(X-X1)-.1 E-10) N 2
X#X1
JUMP 1
2 WRITE (1, F4) A, X,
STOP 3
```

* On “EASE,” an Automatic Programming System for NEAC-2203, by Yasuo Tamura (Electrical Eng. Dept., Waseda Univ.), Toru Kidera (Fuji Electric Co.) and Ken Masegi (Tokyo Shibaura Electric Co.)

** 早大第一理工学部電気工学科

*** 早大第一理工学部電気工学科 (現在富士電機勤務)

**** 早大第一理工学部電気工学科 (現在東芝勤務)

TABLE

. S	
0 1	0 0 0 3 X
0 2	0 0 1 2 X
0 3	0 0 0 0 X
. D	0 0 1 6 X
. T	0 0 2 0 X
A	0 0 2 2 X
X	0 0 2 3 X

文章数 10

命令数 32

編集時間

1 st Pass 35 秒

2 nd Pass { 64 秒 (タイプライタ)
40 秒 (高速パンチ)

目的プログラム

ZA

```
7730100 Y 1000000 Y 1100022 X 1900017 X 1100023 X
1000000 Y 5700000 Y 3000022 X 1700023 X 3008802 Y
2000023 X 1408800 Y 3200018 X 1100024 X 3000023 X
2100024 X 1103340 X 3403340 X 2100019 X 4100012 X
1900024 X 1100023 X 4300003 X 1000000 Y 1900016 X
1100219 Y 1300022 X 7730200 Y 1300023 X 7730200 Y
4400000 X 1000000 Y 0016 Z G
000000090202 Z I 070100000000 Z I 050500000000 Z I
040100000000 Z I 4400000 Z E
```

3. “EASE” の statement 通則

3.1. Statement Number (S. No. と略称する)

1~70 の整数を使用する(これを超えると error 1). CODE statement に続く assembler 形式の左命令の前につけることもできる。S. No. に使用する数に順序はない。同一の S. No. を二つ以上の statement につけることは原則としてできない。

3.2. Constant

Source program に直接数値を書くときは, fix. pt. (integer) または fl. pt. (fractional) のいずれかの形式に従う (fix, float の形式を取り違えると error 7). EASE が記憶する容量の制限から constant と

FORMAT statement の数は計 29 個以内とする。

Constant は Program に続いて打出されるが、同じ値になるものはそのうちの一個だけが記録され他のものにも使用される。

i) Integer (fixed point) 11 桁以内の整数とする、小数点はつけない、被乗数・乗数・除数以外に使用される 2,000 未満の integer は命令で作り、constant として記録しない。

例. 0, 15, 105, 9999999999 (最大の数)。

ii) Fractional Number (floating point) 有効桁数(仮数部) 9 桁以内、小数点をつける、指数部は仮数部の後に E をつけてから小数点をつけずに書く、指数部に “-” はつけられるが “+” は書かない (“+” をつけると error 5)、 10^{-51} 以上 10^{50} 未満の数を書ける、仮数部の整数部分、小数部分がないときに書く 0 は書いても書かなくてもよい。

例. 20.0, 20., 0.2, .2, 12.34 E 3 (=12340.0), 12.34 E-3 (=0.01234), 1.E-49 (最小の数), 9999999999. E 40 (最大の数), 0.0, 0., .0, . (いずれも零)。

3.3. Variable

(1) Simple Variable

a) Declared Variable 1 字以上連続した英字で表わす。5 字以上書いても良いが 6 字目からは判読しない (PRESS と PRESSURE は同一視される)、fix. pt. 用に 10 種、fl. pt. 用に 20 種以内の範囲で declare したものを使用することができる (範囲をこえると error 11, declare しない Variable を使用すると error 6)。

b) Undeclared Variable (declare せずいつでも使える)。

(i) IX 1, IX 2, IX 3 それぞれ Index Register の 1, 2, 3 を意味する、4 桁以内の正の Integer を格納することができる (負数はその絶対値が格納され、4 桁以上の数を入れても下 4 桁だけはいれる)、Sub-routine への Jump 等に Index 3 を使用しているので IX 3 は原則として使用できない、LINK・END などは Index 2 を使用しているので、これらの前後で IX 2 は原則として使えない。

(ii) ⑧, ⑨. 絶対番地を指定する Variable で、⑧ は fix. pt. 用、⑨. は fl. pt. 用である。

例, ⑧ 100, ⑨.100, ⑨ 100 (IX 1) の表現を許す。

(iii) PREV 命令を省略するための Variable、直前に行なわれた () 内、statement の計算結果が

入っている register (ACC, MQR) を示す。

例 1. A #.....≡

B # PREV ≡ (B # A)

C # PREV ※ B ≡ (C # B ※ B)

例 2. A # SINF(PREV) + (B + C) ≡

(A # SINF(B + C) + B + C)

A # SINF(PREV) + B + C ≡

(A # SINF(B) + B + C)

なお -PREV, ABSF (PREV) などは誤用である。

(2) Subscripted Variable declared variable と ⑩, ⑪. には subscript をつけることができる。subscript は fix. pt. の値を示すものでなければならない (TRUNF, ROUNDf を使えば fl. pt. のものも使える)。

例. A (TRUNF(B + C)), A (ROUNDf(B) + I), ただし、B, C は fl. pt., I は fix. pt. の Variable。

Variable statement で dimension を n 個と declare すれば、subscript は 0 から $n-1$ まで使える。それを越えて subscript をつけると他の Variable 用の location を侵すことになる。

Subscripted Variable は subscript の順に location が割当てられる。subscript は原則として () の中に入れるが、subscript が integer 1 個だけのときは () を省いてもよい。

例. A (0), A 0 および A は同内容。

F または F で終る Variable に直接 () をつけることはできない。Function と誤まり error 9 となる。ただし、F 0 (I × 3), TAF 3 (B + C) などは構わない。

いわゆる array などの表現は許されない。

subscript の () の重数、長さに制限はない。

例. A (I (J (K + 3))), A ((I - 1) ※ N + (K - 1)) ただし、I, J, K, N は fix. pt. とする。

⑩, ⑪. の subscript は program をこわさないよう慎重に選ぶ必要がある。

3.4. Function

(1) Subroutine tape によるもの。Compiler 専用の subroutine tape を target program と併用することにより、以下の表現を使用できる。

SQRTF(X) (= $\sqrt{X}$) SINF(X) (= $\sin(X)$)

COSF(X) (= $\cos(X)$) ARTANF(X) (= $\tan^{-1}(X)$)

SINHf(X) (= $\sinh(X)$) COSHF(X) (= $\cosh(X)$)

LNf(X) (= $\log_e(X)$) LOGF(X) (= $\log_{10}(X)$)

EXPf(X) (= e^X) POWERF(X, Y) (= X^Y)

ただし、X, Y は fl. pt. の表現であること。PO-

P5, N6, ¥

A-B ≥ 0 なら S. No. 5 の statement へ、

A-B < 0 なら S. No. 6 の statement へ行く、

例 2. IF(SINF(X)) N5 Z6 ¥

SINF(X) > 0 なら次の statement へ、

SINF(X) = 0 なら S. No. 6 の statement へ、

SINF(X) < 0 なら S. No. 5 の statement へ行く、

(4) ITER statement

ITER([S. No.])[Variable], [Variable or Const.],
(変数) (刻み幅)
 [Variable or Const.], ¥
(上限)

の順に書く。

3 個の variable を順に IJK とすると、

ITER (5) I, J, K, ¥ は $\begin{cases} I \# I+J & ¥ \\ IF(I-K) Z5 N5 ¥ \end{cases}$

を意味する。

I の初期値は Loop の前で与えておき、この statement は Loop の最後に置く。

I # [Initial value] ¥
 5 ¥
 ¥
 ITER (5) I, J, K, ¥

上の例で I が IX1, IX2 または IX3 のいずれかで、K が 1 の場合に限り J が -1 であってもよい。

例. ITER (5) IX1, -1, 1, ¥

この場合は IX1 が 0 になったとき、iteration は完了し S. No. 5 の statement に戻らない。

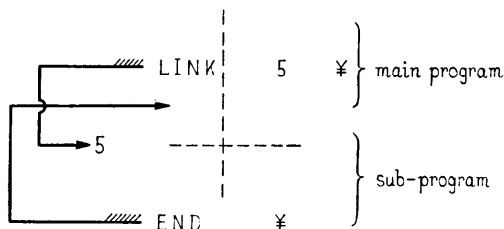
(5) LINK Statement

LINK [S. No.] ¥ と記し、Set index Jump をする。Index register No. 2 を使う。この LINK は END と組合わされて、Source program の一部を Subroutine として使用する際に用いられる。

(6) END statement

END ¥ と書く。LINK で jump してくる routine の最後などにおく。END があると index register No. 2 に記録してある位置に戻る。

例



END は Runge Kutta Gill の subroutine を使った場合などにも使用する。

(7) S.R. statement

パラメタ数の多い subroutine と主プログラムを接続する statement である。

SR [address] ([parameter], [parameter], ..., ...) ¥ の形式で書く。

SR の次に書く address は subroutine の格納開始番地で target program はここへ Set Index (3) Jump して SR statement の次の statement に subroutine から戻ってくる。パラメタは subroutine によって個数、配列が異なる。Variable をパラメタとして使用する場合、() を伴う subscript のあるものはいけない。

例 1. SR 1000 (5, A, B) NEAC-2203 の Library routine の F 項 (Matrix 関係) は大抵この形式で書くことができる。

例 2. SR 500 (N, .S5, H, X, Y, DY, Q) Runge-Kutta Gill の subroutine と結びつくときの形式である。

4.3. Declaration statement

以下に述べる declaration statement は命令は作らないが、編集上必要な情報を与える。

(1) VARIABLE statement

Source program は必ずこの statement から始まる。

VARIABLE

FLOAT, [Variable], [その数], [Variable], [その数], ..., ¥

FIX, [Variable], [その数], [Variable], [その数], ..., ¥

の形式で書く。

Variable は subscript のない形で書く。Variable の数は最大の subscript に 1 を加えたものを書く。subscript をつけないで使うものには、1 と書く。

FLOAT, の後には、fl. pt. の Variable を 20 種以内の範囲で列挙する。FIX, の後には fix. pt. の Variable を 10 種の範囲内で列挙する。fix. pt. の Variable がいない場合には FIX, を省いてよい。

program 中にこの statement は書かない。

例. VARIABLE

FLOAT, TEMPERATURE, 3, PRESSURE, 4, FIX, COUNT, 5, ¥

(2) FORMAT statement

FORMAT ([integer], [integer], [integer]) ¥
の形式で記す。

FORMAT statement には必ず S. No. をつける。
この statement により、WRITE statement の出力
形式が指定される。3 個の integer は左から、打出
桁数 fl. pt. のときは仮数部桁数); data 間の space
数; 1 行当りの data 数を示す。

FORMAT statement は 1 個の data として記録
される。

例. 6 FORMAT (9,2,3) ¥

(3) CODE statement

「CODE ¥」と書くことにより、その後に assem-
bler 形式で最大 50 個の命令 (mnemonic code) を
つけることができる。assembler 形式の終りに一括
して「¥」を 1 個つける。assembler 形式は左命令か
ら始まる。mnemonic code は第 1 表のとおりである。

Index の 0.1,2,3 は必ず記す。Address は Vari-
able (subscript は () のないものだけ) あるいは
.S[S. No.] あるいは数字で直接書く (絶対番地)。
Address の終りに「,」をつける。

FON, FOF の命令を使ったときは、次の statement
を Mode statement にする。

例 1. CODE ¥	例 2. CODE ¥
CHR 0 1003,	⋮
HRS 0 2,	FON 0,
UCJ 2 .S5,	⋮
	FOF 0,
	⋮
NET 0, ¥	, ¥
	FLOAT ¥ (FON の
	命令を作る)

4.4. Input-Output statement

(1) READ statement

READ([機種指定]) [Variable],..., ¥ の形式
で書く。

subroutine を使用しているので、target program
に subroutine tape を添附する必要がある。

Photo reader 1 号機を使うときは 1, 2 号機のと
きは 2, Key board からのときは K, Mechanical
tape reader のときは M, をそれぞれ機種指定とす
る。

data は前出の Constant の形式に従うこと。

例. READ (1) X (I), Y (J+I), K, K (I+J), ¥
15.5 E3, 0.2, 50, 3 などと punch された tape を
PTR 1 号機から読み込んで、それぞれの Variable
の新しい値とする。

(2) WRITE statement

WRITE ([機種指定], F[S. No.]) [Variable], ...,
¥ の形式で書く。

機種指定は 1, 2, 3, 4 があって、それぞれ print,
punch, print and punch, high speed punch を指
定する。

この statement も subroutine がある。

F の次に FORMAT statement を S. No. で指示
する。F および S. No. を省略すると前の指定を採用
する。この statement で打出した data は READ
statement で読み込むことができる。

例. WRITE (3, F3) X, I, ¥

(3) AWRITE statement

AWRITE の後 ¥ までを印字する。¥ は打てない。

例. AWRITE EASE ¥ と書けば EASE と印字
する。

4.5. Mode statement

“EASE” は Jump する先の mode について考慮
していないので、Jump する前または Jump する先
で mode Statement を適当に書き加える。EASE は
加減乗除のない限りできるだけ現在の mode のまま
で compile するが、s. No. のつく statement は皆
mode statement にすれば間違いはない (FORMAT
は別。SR の前にも書く方が無難)。“EASE” は FOF
の状態から compile を始め、直線的に mode を変化
維持するものとして FON, FOF の命令を作ってい
る。

(1) FIX statement FIX ¥ と書く。FOF の
命令を作る。

(2) FLOAT statement FLOAT ¥ と書く。
FON の命令を作る。

4.6. Break Point statement

BKP ¥ と書く。BKP の命令を作る。

5. 使用法

5.1. EASE の入れ方

EASE は initial tape と同様の操作で入れる。
manual で 12 桁読み込んで、0000 番地に格納し、
0000 番地から start させると自動的に全部入る。読
み込みは 2 分少々で終る。

5.2. Source program の扱い方

Source program を punch した tape を PTR
にセットして restart (100 番地) させると、compil-
ing の 1st pass を始める。このとき Break point

第 1 表 Mnemonic Code Table

00		COO	50	Left Shift	LST
01	Store Temporary Memory 1	ST 1	51	Character Left Shift	HLS
02	Store Temporary Memory 2	ST 2	52	Right Shift	RST
03	Load Temporary Memory 1	IT 1	53	Character Right Shift	HRS
04	Load Temporary Memory 2	IT 2	54	Shift And Count	SAC
05	Equal Jump	EJP	55	Round Off	RDO
06	Ad Extract	ADE	56	Table Look Up	TLU
07	Compare Jump	CJP	57	Floating On	FON
08	Clear Add Extract	CAE	58	Floating Off	FOF
09		CO 9	59	Normalize	NOR
10	No Effect	NET	60	Read In	RDI
11	Store	STR	61	Type Special	TSP
12	Store MQR	SMQ	62	Clear Character Read In	CHR
13	Load MQR	IMQ	63		C 63
14	Load MDR	IMD	64	Type Out	TYO
15	Character Load MDR	HLD	65	Character Type Out	HTO
16	Character Bring	HBR	66	Clear Read In	CRD
17	Divide	DIV	67		C 67
18	Clear Store	CST	68	Read Card	RDC
19	Clear Character Bring	CHB	69	Write Card	WTC
20	Add	ADD	70	Raise Index	RIX
21	Subtract	SUB	71	Lower Index	LIX
22	Plus Multiply	PML	72	Memory to Index	MIX
23	Minus Multiply	MMU	73	St re Index	STI
24	Add Absolute	ADA	74		C 74
25	Subtract Absolute	SUA	75	Index Zero Jump	IZJ
26		C 26	76	Index Non-Zero Jump	INJ
27		C 27	77	Set Index Jump	SIJ
28	Raise	RAS	78	Reard Card Conditional	RCC
29	Lower	LOW	79	Index to Accumulater	INA
30	Clear Add	CAD	80	Load Core Block	ICB
31	Clear Subtract	CSU	81	Load Core (Load Buffer)	IDC
32	Clear Plus Multiply	CPM	82	Read Drum	RDD
33	Clear Minus Multiply	CMM	83		C 83
34	Clear Add Absolute	CAA	84	Store Core Block	SCB
35	Clear Subtract Absolute	CSA	85	Store Core (Store Buffer)	STC
36		C 36	86	Write Drum	WTD
37		C 37	87	Offering Jump	ORJ
38	Clear Raise	CRA	88	Set Core (Set Buffer)	SFC
39	Clear Lower	CLO	89		C 89
40	Plus Jump	PJP	90	Read Tape	RDT
41	Minus Jump	MJP	91	Write Tape	WTT
42	Zero Jump	ZJP	92	Write End Mark	WEM
43	Unconditional Jump	UCJ	93	Slect Tape Mark	STM
44	Stop Jump	SUJ	94	Jump Tape Mark	JTM
45	Overflow Jump	OFJ	95	Forward Feed	FFD
46	Break Point	BKP	96	Reverse Feed	RFD
47	Select MDR	SMD	97	Rewind	RWD
48	Jump if 8 in MDR	JP 8	98		C 98
49	Busy Jump	BJP	99	Search	SER

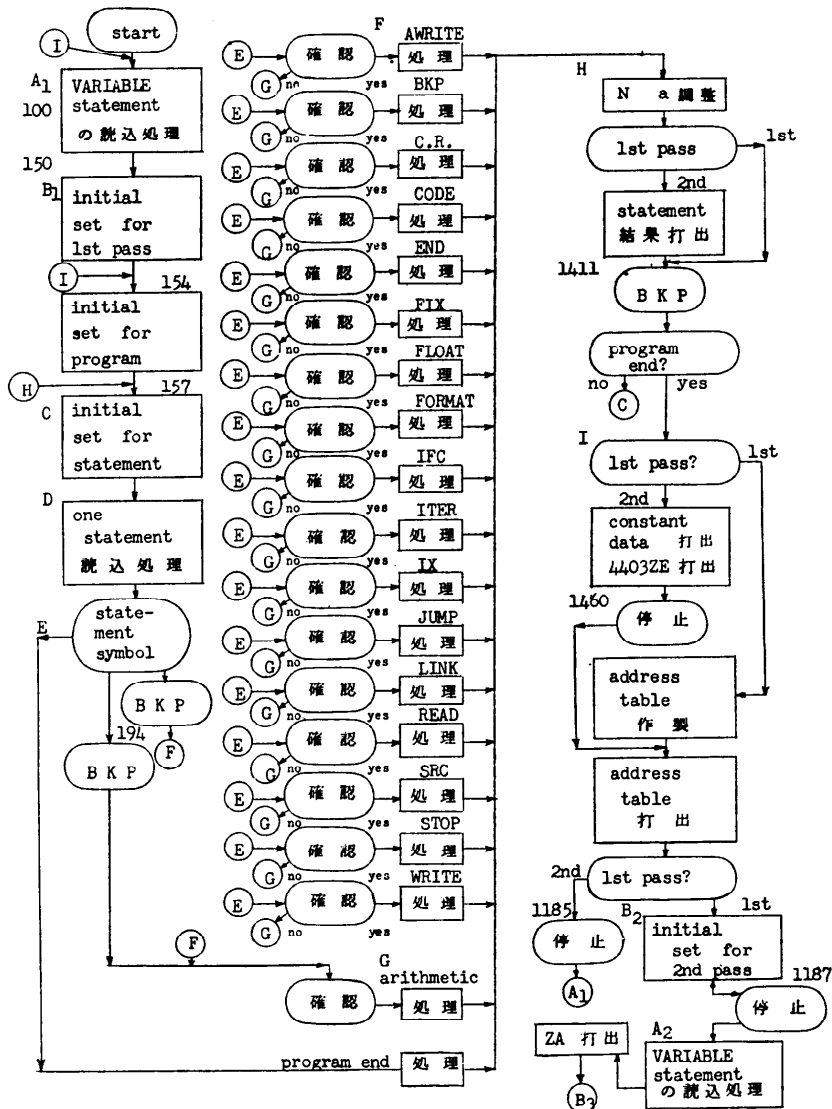
は OFF にしておく。

読込処理が終ると、“TABLE”と打出して停止する。打出機種を変更するとき以外は tape feed して restart させると、S. No. とその番地、constant の格納開始番地、作業番地、fix. pt. の variables と開始番地、fl. pt. の variables と開始番地を打ち出して来て停止する。これで 1st pass が完全に終わったことになる。なお、このときの作業番地の開始番地を示

す数が命令の格納番地個数を示す。

1st pass が終わったところで、source program を PTR にもう一度セットして tape feed しておいてから restart させる。これからが 2nd pass である。source program を読み込みながら target program を打出して来て終ると、停止する。

この後、tape feed しておいて restart させると再び“TABLE”と打出して来て停止するから、restart



第 1 図 “EASE” compiler のフローチャート (全体図)

させるともう一度 table を打出して来て compiling が完全に終る。別の source program を PTR にセットして restart させると、また 1st pass から始めるわけである。

以上の操作により target program が得られるわけである。ある程度操作に変化をつけることができるが、ここでは省略する。

5.3. Error の処理

Source program に文法上の誤りが発見されると、

ERROR NO. と誤りのある statement を打出してくる。ERROR は次のような番号で分類されている。

1. S. No. に 71 以上の数を使ってあるが、constant を statement の初めに書いてある (ex. 20.0# ……等)。bonstant および FORMAT statement が 30 種以上ある。その他。

2. 1 statement 中に 91 字以上があるが「等」が脱落している。1 statement に 15 個以上の作業番地がある。1 statement に 51 個以上の命令ができた。

その他.

3. 何か脱落しているものがある. その他.

4. 余計なものがある (二重の operator など).
その他 (3 と 4 は見方次第でどちらにもとれるものが多い).

5. Constant に誤りがある. その他.

6. Variable に誤りがある. その他.

7. fix. pt. と fl. pt. の使い分けが十分でない.
function に「F」が脱けている. その他.

8. Mnemonic code に誤りがある [Code statement の中で]. その他.

9. Function の名称を誤まっている. F で終る Variable に () のある subscript をつけた. Function の Argument の mode を誤った. その他.

10. その他各 statement で要求される書き方に従っていない.

11. Variable の declaration に誤りがある.

「,」の脱落, 数を書かない, FLOAT を書かない. Variable の個数が多過ぎる, などが多い. なお, Variable statement の error のときは全文を打ち出さない.

以上簡単に “EASE” compiler の紹介をした. 第 1 図に FLOW CHART (全体図) を掲げる.

6. 結 言

昭和 35 年 1 月に Block Diagram Simulator “SWEEP”¹⁾ と称する特殊自動プログラムを LGP-30 用に作製してから, 是非一般目的の compiler を作りたいと望んでいたが, ひとまず NEAC-2203 用の “EASE” に落着いたことになる. EASE の作製に本腰を入れたのは昭和 36 年 9 月なかばであるから, 約 6 カ月余りを費した勘定である. 作製途上において痛感したのは key の種類の不足であった. 計算機の命令構成およびコンソールなどの機能にプログラミング側の要求を反映させることは今後の問題の一つであろう.

EASE の作製に当って, 多大の計算時間を割当てて下さった電子計算室長難波正人教授はじめ関係各位に厚く御礼申し上げる.

参 考 文 献

- 1) 田村: ブロックダイアグラムシミュレータと自動プログラム方式の結合. 電気学会誌, 昭和 35 年 7 月号 pp. 923~930.

(昭和 37 年 5 月 11 日受付)