Automatic Coding について（I）

— Assembler —

穂鷹良介

1. ここでいう Automatic Coding とは，電子計算機のための一種の飜訳プログラムのことで，いわゆる Assembler, Compiler 等がその例である。つまり，或程度人間の言葉に近い言語で書かれた問題等を computer が直接理解出来る機械語に飜訳するためのプログラムである。更にいい変えるならば，高級な言語（例えば人間にとって分かり易い形で書かれた言語）をより低級な機械語に変換するプログラムである。

計算機は普通，ごく少数の（30から300位）基本的な命令を行えるように設計されている。従って，通常の問題がそれ程単純でない以上，個々の問題を計算機で解くには当然，もとの問題をそれらの基本命令でとり扱える範囲まで分解しなくてはならない。勿論普通の問題を直接受けつけることの出来るようにも，計算機自体をあらかじめ作ることも理論上は可能であろうが，種々の用途を考えるとき夫々に適した操作が出来る計算機というのは，無限に多くの機能を持たねばならぬであろうから，経済的に引合わない。むしろ問題夫々に応じて，基本命令を適当に組み立てて，それによって，その問題に適した機能をその都度もたすようにした方が賢明である。従って，現在の電子計算機を使用する場合には，そのたびごとに，基本命令の組み立てを行わなければならないのであるが，多くの場合，これが仲々に laborious な仕事である。Automatic Coding はこの過程をより容易に行うために計算機にある種の仕事を分担させるためのプログラムといえる。

Automatic Coding にも色々の段階があって，単に機械語を他の分り易い言葉におきかえただけのものから，複雑な数式を一気に一連の比較的長い機

械語におきかえるものがあり，すでに両者とも実用の段階に入っている。ここでは前者のいわゆる Assembler といわれる段階に属する Automatic Coding を，その作り方，理想等について，筆者が昨年作り上げた HAP (Hotaka Assembly Program) を例にして詳細に紹介する。もし計算機の基本的な構成が同じであるならば，以下にのべる HAP の説明をもとにして，即座に Assembler の coding が出来る筈である。事実筆者は，以下にあげる Flow-Chart を見ながら，即座に HAP の coding を行って完成した。また，基本構成が異なる計算機においても，流れ図の大体の構成が若干の修正で通用するから，他の計算機においても，以下の図式から，HAP と同様の assembler を作ることは容易である。

2. HAP は38年8月に開発を始め，同年10月には実用の段階に入った。Coding, Debug とも筆者によってなされた。以来幾多の問題を HAP language によって Coding したが Debugging も比較的容易で，論理的な誤りもないので，以下の HAP の Flow Chart は信用するに足る。HAP が使われる機種は，OKITAC 5090型である。HAP の特徴を列記すれば次の通りである。

(1) Instruction Code は machine と同じく数字の2桁である。

これは，いわゆる assembler の資格からは望ましくないことであるが，computer の性能を十分に発揮させる，凝った program 等を新しく開発するときは，single order または tracer 等で debug する必要が多大であるため，operation code は直接 machine code としておぼえた方が，かえって容易であるという筆者の経験からこのようにしただけのことで，普通の alphabetical な code を受けつけるように assembler を変更することは，差程難かしい問題ではない。(1) 実際に使って見ると決して悪くない idea であ

(1) 後に alphabetical な symbolic code をも受付けるように改良した HAP を作ったが，その変更は全く容易に出来た。

った。

(2) One Pass である。

core memory が 4000 ある計算機で one pass であるのは，別段めずらしくもないが，とにかく長いプログラム（1000語をこすようなもの）になると source program（もともとのプログラム，つまり，assembler で飜訳しようとするプログラム）を一回通すということも面倒なものであるので，one pass 方式は two pass に比較して大きな利点である。

(3) Actual 使用を許す。

address 部に直接，数字，文字を書き込むことを許した。しかし one-pass のためと，core memory 節約のため，Actual の Data は固定番地から格納されていく方式になっている。

(4) 浮動小数点入出力 sub-routine の内蔵

科学計算につきものの浮動小数点演算を容易にするため，ある程度 Format の定った浮動小数点数値は，プログラム実行中にも HAP を使用することにより容易に出し入れすることの出来るようにした。

(5) 記号番地の重複を可能にした。

sub-routine をやはり HAP 言語で書いた場合，それが本来のプログラムのことを考慮していなければ，当然一つの label に対して二つの異った番地を割り当てる可能性が生じる。HAP ではこのことを避けるために sub-routine の HAP で書かれたものを読むときには，元の label を一時忘れるようにしてある。sub routine 等を読み終った後にまたもとの記号表を想い出せよいのである。この考え方は ALGOL 等の compiler を作るときに own variable の関係で重要な役を果すように思われる。

(6) 重複した label も同じ番地が割当てられる場合には許した。

このことにより，プログラムを 2 回通して 2 pass と同じような便利さも利用出来る。逆に記号表が正しいか否かの check も出来る訳である。

(7) 入力機種の選定を或程度自由にした。

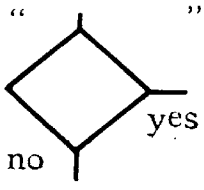
OKITAC は，Data を入れる前には，必ず入力機種を選定する仕組みになっているが，HAP では，それを SW の切りかえで manual 操作も出来るようにした。これは debug のときに便利である。

以上思いつくままに HAP の特徴をあげたが，細かい点はまだあげきらない。しかし，それも以下の Flow-chart を慎重に検討すれば分ることなので省略する。なお HAP 自体は 1000 word のプログラムで現在使っているが多少の不便をしのぶならば 500 word 内外に押えることが出来るから core memory が多少少くても使える。

3. 以下 HAP の Flow Chart をあげ，必要な comment を書きそえる。図式中，若干特別な記号を使用するので，それを説明する。

"＝" 等号はその左辺と右辺のものの関係を示す。ALGOL のものと同じ意味である。

":＝" これも ALGOL の記号の意味で，左辺のものが右辺のものでおきかわることを意味する。



" " 条件判定 box　この box の中には relational operator "=" ">" 等で複数個のものを判定している。条件の満足された場合には yes と示された流れに進み，さもなければもう一方に進む。

"(**LABEL**)" 左のように label を（ ）でくくったときには，その label で定義されている番地の内容を示す。label は絶対番地でもよい。

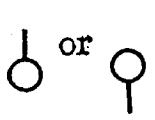
"〔**LABEL**〕" 左のときには，label が定義されている番地そのものを表示する。label は絶対番地そのものでもよい。

例えば A を 300番地と定義しておいて，300 番地に ＋500 が入っていれば

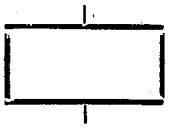
（A） は ＋500 で 〔A〕は 300 である。

(300) は ＋500 で〔300〕は 300 である。

この 2 つの区別は計算機では重要である。

“ 

” connector　Flow chart の離れた部分をつなぐ記号，丸の中には記号もしくは数字が入る。

“ 
” 操作 box　この box の中で種々の操作を行う。

なお，各 box の横には label のあることがあるが，それは，coding の際そこに対応する label と考えて良い。

OKITAC 5090A の1語の構成は符号1bit（正なら0，負なら1の bit がつく）と12桁10進の数値で，1桁は4bit である。またこれを binary と見て 48 bit と符号と考えることも出来る。

Register としては 10進4桁の index が1個，Accumulator として UA (upper accumulator)，LA (lower accumulator) が各1個 及び SCC が，この assembler に関係する。

入力命令は特に注意を要する。たとえば，数値をよみこむ命令で，何個よむかを指定すると，数値がその個数に充たないときは，UA の LSD（後）にその数値をいれ，数値以外のものは皆エンド・マークとして LA の LSD に入る。数値の個数が読み込みの個数に等しいときには UA にその個数だけが，後(シリ)から入るだけで LA には何も入らない。また UA は12桁であるがそれ以上読み込んだときには，UA の上(カミ)からそれらの数値は失われていく。

英数字読込み（Read Alphanumeric 略して RAN）も同じ原理で，英数字以外のものを上と同じ要領で LA に検知する。

4. HAP の文法について述べる。

i) Control Code　これは HAP に対する命令であって，source program には書かれても，object program には原則として対応する instruction が植えつけられない命令である。

a) HAP.　記号番地及び actual data を凡て解消する。

b) LO (A±n)　ここで A は label，n は絶対番地である。A を書かない

場合は単に LO (n) と書く。label はそれまでの assemble の過程で定義されているものでなくてはならぬ。この命令は以下に読く instruction をその label もしくは label と絶対番地の address 演算の結果の番地以下に格納すべきことを示す。

c) START (A±n) ( ) 内は b) と同じ性質のものである。この命令を assembler が読解すると今までの assemble の結果，未定義の label がないかどうかを調べ，ないときに，A±n 番地に jump して止る。もし未定義な label があるときには，その旨印字して，assembler の progran 読み込み番地に jump して止る。

d) HALT. この control code によってHAPは読み込みを一旦中止する。

e) END. この指令により，今までの label に割当てられた番地を印字する。未定義 label についてはその旨を印字する。

f) PUNCH ($A_1 \pm n_1$, $A_2 \pm n_2$) ここでの addrees演算は b) で述べた性質のものでなくてはならない。この指令により，core の内容を self loading の形で $A_1 \pm n_1$ 番地から $A_2 \pm n_2$ 番地までを紙テープに punch out する。この self loading のテープの始めには Boot Strap がつけられ，何等他の入カルーチンなしに，読み込み check を行いながら，再び読みこむことが出来る。なお，OKITAC 5090 は普通，Data を10進として扱っているが，この punch out された tape は core の内容の凡ての bit を忠実に punck out するから，2 進 data もこわされることがない。

g) ENTER. 及び EXIT. ENTER. を読み終ると HAP. は今までの定義表 (150ヶ分) を全部一旦別の場所に移しかえる。EXIT. の命令により，前に移した定義表を再び戻す。この際 actual data には何も変化がない。従って ENTER. の後には，今まで Source Program においての定義表の一部と重複しても double defined にはならない。また当

然前の定義表は忘れている。[2]

h) JUMP（A±n）　この（　）内の address 演算は b) に対する同様の意味において有効である。この命令によって HAP は今まで assemble した内容が右命令に止っているような場合（OKITAC 5090 では Paired order である。）右命令に non effect の instruction である 69 をつめて，A±n に jump してしまう。

ii) Pseudo Code　ここでの pseudo code とは，本来の計算機にはない命令であるが，HAP. が適当に解釈して，あたかも 1 つの命令として使えるように assemble する命令である。

a) RST（A±n）　（　）内の address 演算は i)−b) と同様の意味と制約をもつ。この pseudo code により，それまでに instruction が左命令のみの翻訳で終っているときには non effect の instruction を右命令につめて，それ以後に A±n word だけ数値の +0 を格納する。

b) READ 1.　この pseudo code を書いておくと object program 実行の際には次のことを行う。

イ) index=0 のとき，Fortran READ 1 type で書かれた data を 1 個だけ PTR より読みとり UA に格納して次の stop に進む。なお，READ 1 type とは，例えば

23.0　+5.4　−2.　+0.　−.0　5.

の如き Format で書かれた data で data と data の区別は space か C_r/L_f（改行復帰）でなすものである。

（2）ENTER or EXIT なる control code を使用しないプログラムの際には記号番地の table が倍だけ遊んでしまうことになる。この無駄を省くために table の裏表を連続した記憶装置（core memory）に並べておいて，もし表（おもて）の table の記号番地の容量を越えて label が使用された場合には，そのまま裏の table に記憶するようにし，その代り，裏表を使い分ける ENTER. EXIT 等の controle code は受付けないようにすることが考えられる。このようにすれば，table の容量は 2 倍になる訳である。この試みも HAP に対してなされた。

ロ) index≠0 のとき，やはり Fortran の READ 1 type の data を読むのであるが，今度は data の後に "S" が来るまで data を順に index の示す番地より格納する。

（なおこの読み込み routine は OBM の高橋喬氏が作成したものを HAP. に組みこませて頂いた。）

c) LPR. /n. /A±m. このように source program に書いておくと，program 実行の際，A±m 番地より n 語を floating の data として line printer で打ち出す。n はまた記号番地も許す。

iii) Instruction Code, Address.

HAP. では instruction code は10進 2 桁の数値で machine code と同じくそのまま書き "/" で区切る。その後には address 部を書き込む。例えば

23/3000.　　69/A0.　　24/A+5.

index modification は address の後に ,X. を書きそなえることによってなされる。例えば

23/B0,X,　　01/,X.

label が定義されていない時には address 演算は one pass では出来ない。但しControl code の HAP. をよまさずに program を 2 度通すと可能である。

iv) Location

label は 4 文字以内の英字で始まる英数字が許される。それらの定義には次の 2 通りの方法がある。

例　A=125)　　B=A+60)

の如く直接定義するかもしくは instruction または data の前または後に

A;　　B;

のように書くと以下のプログラムが格納される番地にAまたはBが夫々定義される。なお特に定義しなくても UA=8000) という定義を HAP は常にお

ぼえている。

v) Data

HAP. は次の3種類の data を許す。

イ) 数値 data を MSD につめるとき

例　+521/　　−312/

の如く符号の後に12桁以内の数値を書き“/”で区切る。

ロ) 数値 data を LSD につめるとき

イ)と同じ要領であるが end mark は“.”である。例 +2.　+0.　−6.

ハ) 文字 data

*の後に6文字書く。

例　*ABCDEF　　*012)A.

vi) Actual Data

address 部に上の3様の data の書き方が許される。しかし，実際の data は固定番地より格納される。

例　HAP.　　LO (140)　　02/+4.　　69/+4.

と書くと assemble された結果は固定番地を2999番地とすると

140番地→+02　2999　69　2999

2999番地→+00　0000　00　0004

のように assemble される。

vii) ERROR MESSAGE

以上の文法にあわないときには夫々次のような error message を打つ。

イ) UNDEF　未定義 address で演算 A±n を行ったとき又は未定義で START (A), LO (A), PUNCH (A, A), JUMP (A), RST (A) を行ったときは assemble の最中にその旨打ち出して一旦 assemble をやめる。他に END. START (　) の項でのべた場合にも未定義 Label があるとこのmessage を打ち出す。

ロ) W DEF　double defined の記号を読んだときにこの message を打

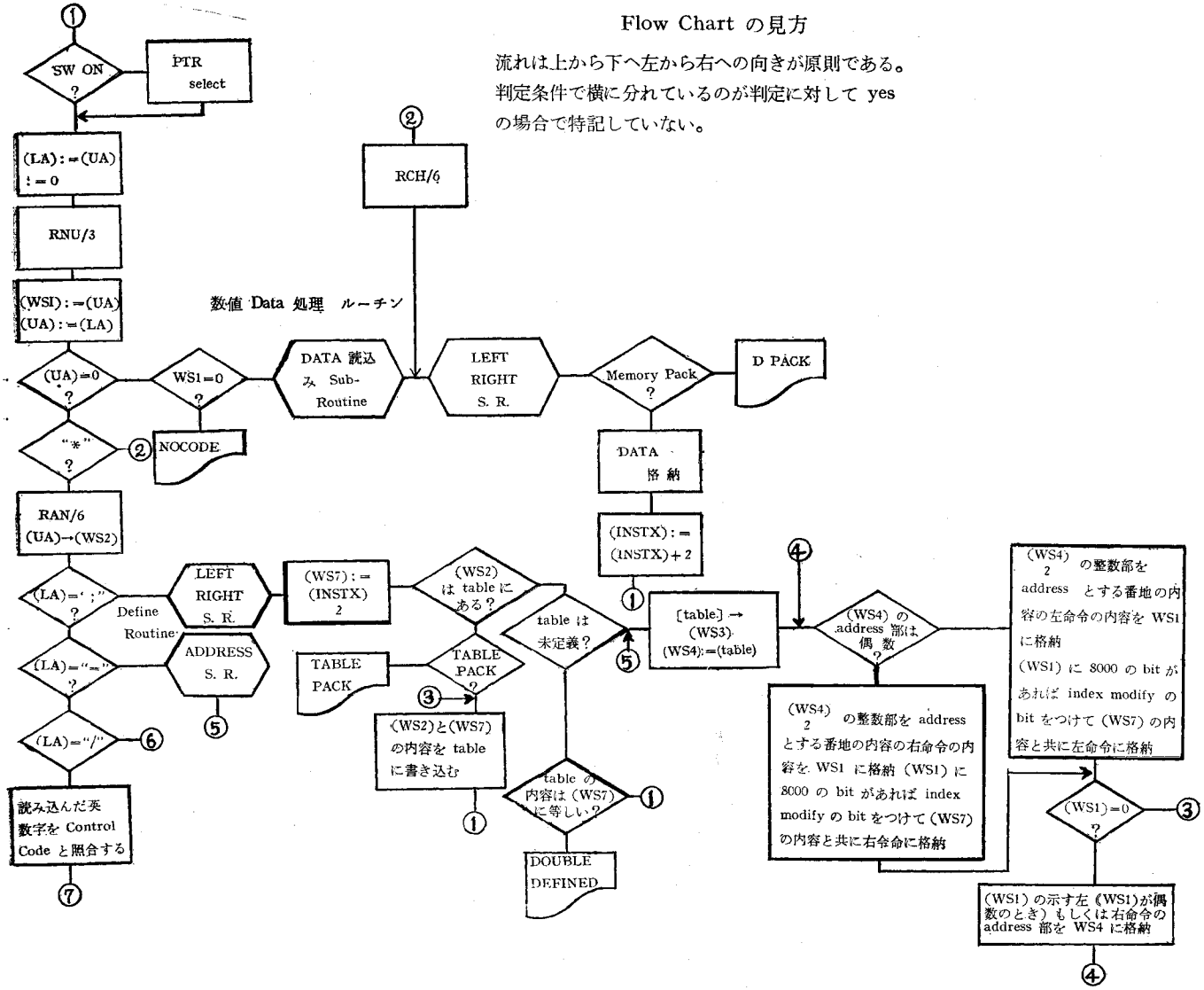
ち出す。なお HAP では，同じ定義ならば 2 度行っても double defined にはしない。

ハ) NOCODE　HAP の文法に合わない program error が生じたとき

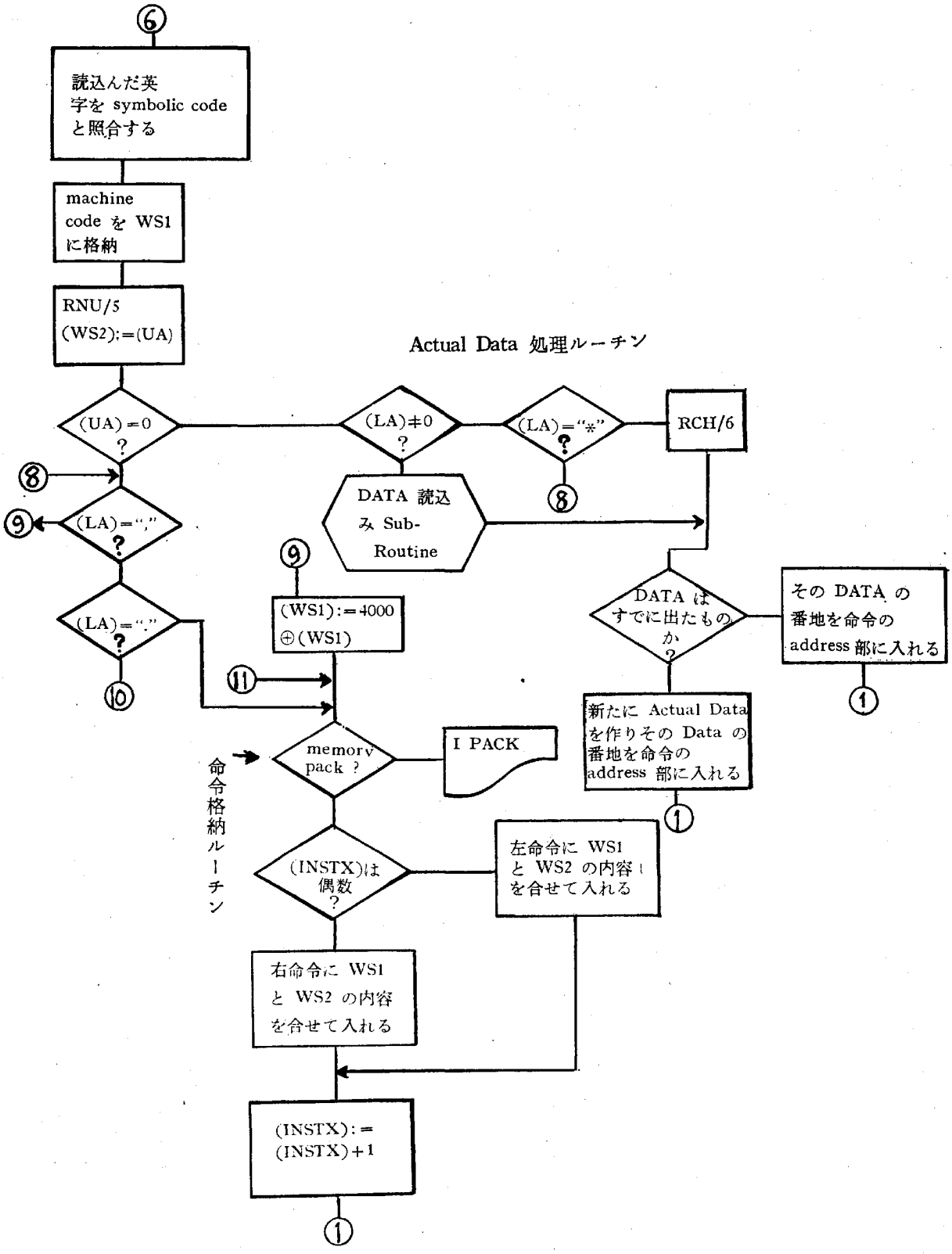
ニ) APACK, IPACK, DPACK, TPACK　いずれも assemble の際の容量を over したときに打ち出される message で，順に Actual Data の場合，Instruction の場合，Data の場合，記号表の場合である。

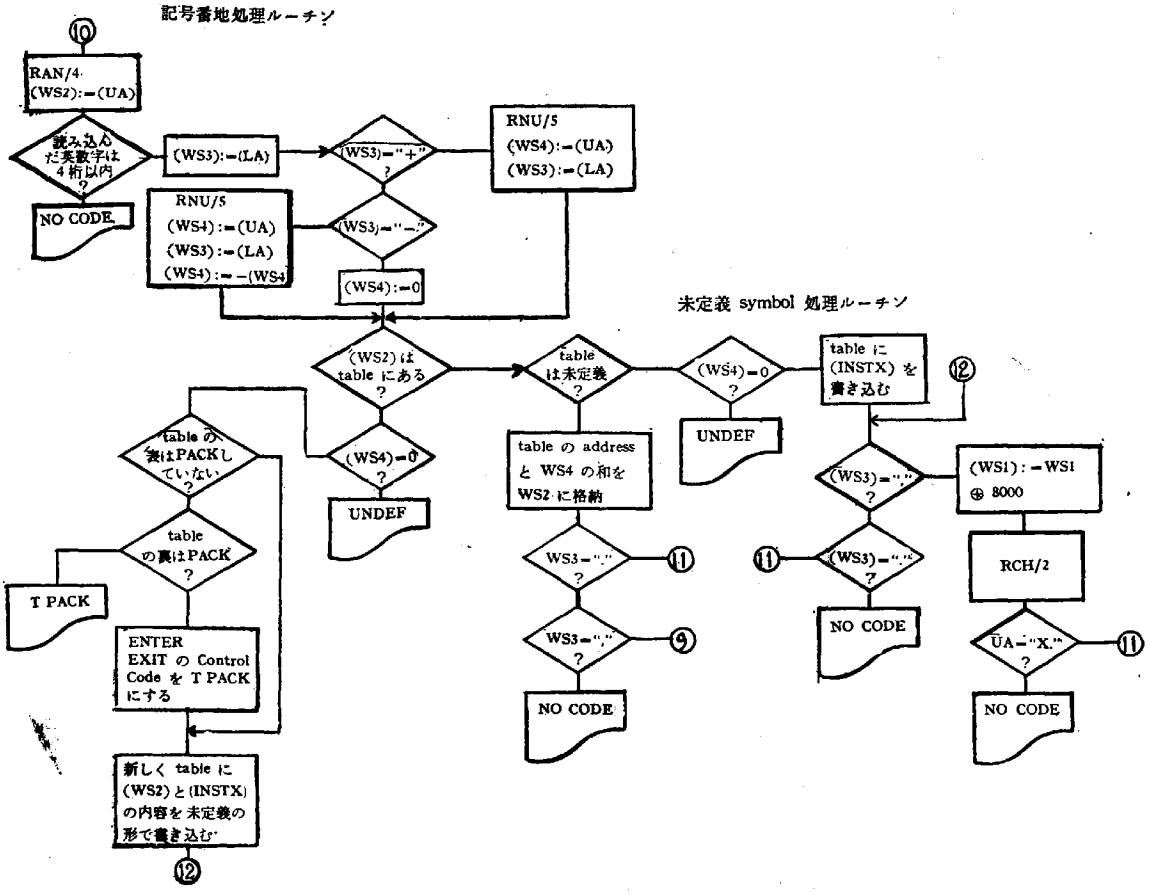
Flow Chart の見方

流れは上から下へ左から右への向きが原則である。
判定条件で横に分れているのが判定に対して yes の場合で特記していない。

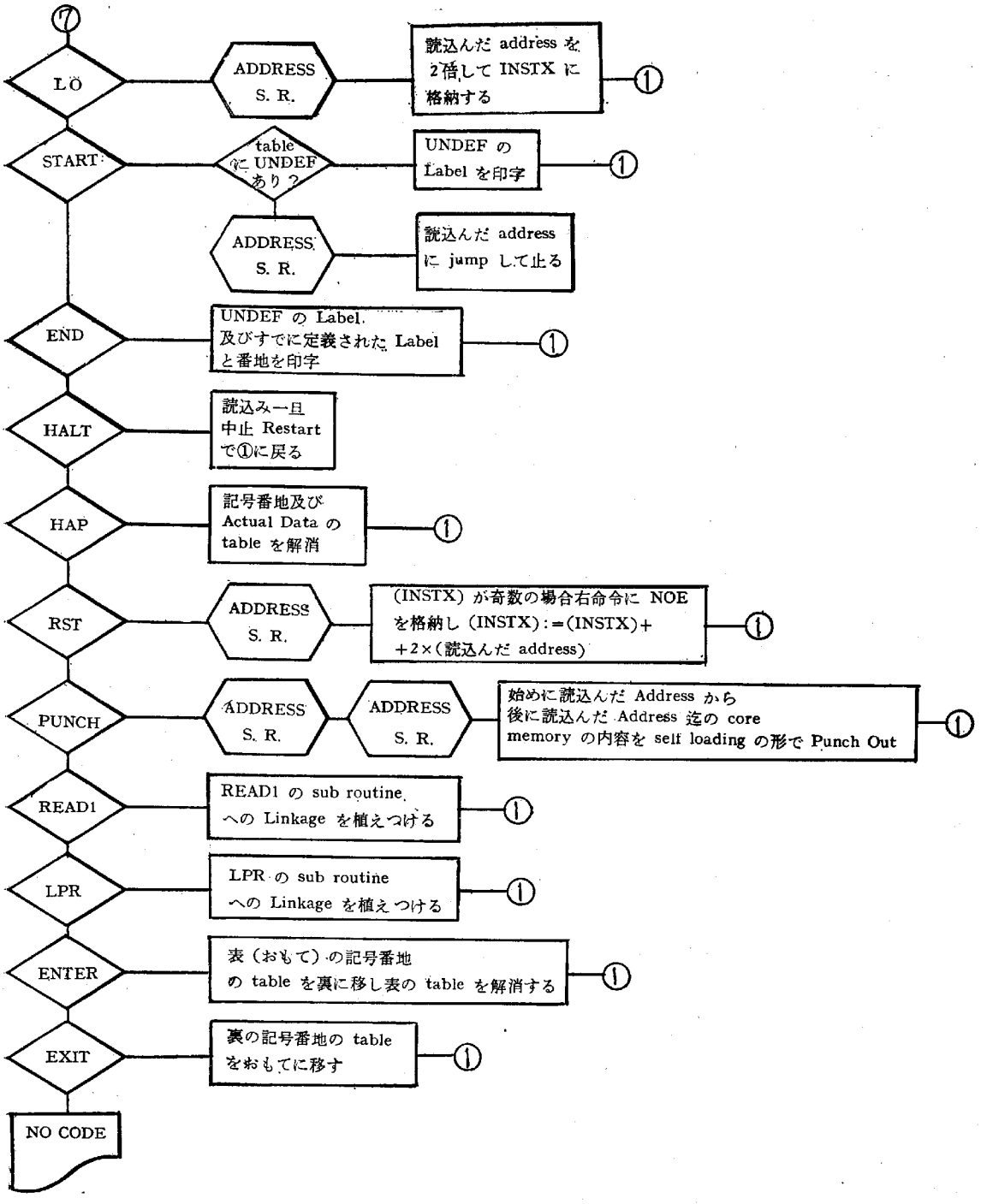


命令作成及び格納ルーチン

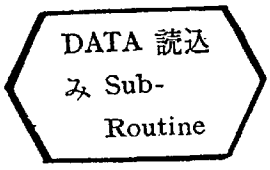



記号番地処理ルーチン
⑩
RAN/4
(WS2):=(UA)
読み込んだ英数字は4桁以内？
NO CODE
(WS3):=(LA)
(WS3)="+"？
RNU/5
(WS4):=(UA)
(WS3):=(LA)
RNU/5
(WS4):=(UA)
(WS3):=(LA)
(WS4):=−(WS4)
(WS3)="−"？
(WS4):=0
(WS2)はtableにある？
(WS4)=0？
UNDEF
tableの後はPACKしていない？
tableの裏はPACK？
T PACK
ENTER EXIT の Control Code を T PACK にする
新しく table に (WS2) と (INSTX) の内容を未定義の形で書き込む
⑫
未定義 symbol 処理ルーチン
tableは未定義？
tableの address と WS4 の和を WS2 に格納
WS3=","？
⑪
WS3="."？
⑨
NO CODE
(WS4)=0？
UNDEF
tableに(INSTX)を書き込む
⑫
(WS3)=","？
⑪
(WS3)="."？
NO CODE
(WS1):=WS1 ⊕ 8000
RCH/2
UA="X."？
⑪
NO CODE

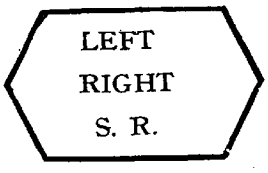
Control Code 処理ルーチン



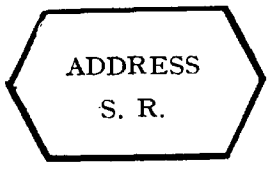
Sub Routine の機能



符号のついた数値 Data を End mark が "/" 又は "." に従って夫々1語の左又は右につめた形で読取る。



(INSTX) が奇数のとき$\frac{\text{INSTX}}{2}$の整数部の示す番地の右命令に NOE の命令を格納し INSTX の内容を1だけ増し常に (INSTX) を偶数にする。



A±n. ここで A は記号番地 n は Decimal Integer を読み込み table をさがしてその値を計算する Subroutine A が undefined の時にはその旨印字して止る。