Automatic Coding について (Ⅱ)

—— COBOL Compiler (2) ——

穂 鷹 良 介

目　　次

3. コンパイラー，通訳ルーチン，モニターの動作

2. ではコンパイラー等の構成内容，仕様について述べたが，3. ではそれらの論理構造，FLOW について述べる。PHASE Ⅳ (PD) の簡単な verb の処理，通訳ルーチンの簡単なルーチン等その動作が自明であるもの，また，すでに動作が 2. で説明されているものについては叙述を省略する。

3. 1. プリコンパイラー

プリコンパイラーの大体の FLOW は以下の通りである。

(1) モニターからプリコンパイラーに制御を移す。

(2) Source Program をカード一枚分カードリーダーより読みとる。

(3) Listing の指定がないときには(5)に行く。

(4) Source Program をカード一枚分 LP に OUTPUT する。

(5) よみとった source program が END Card (Sequence no. が 999999 のもの) ならば，制御を Monitor に戻す。

(6) source program カード一枚分を **2. 2.** で述べたように syllable 化して指定された MT 上に OUTPUT し，(2)に行く。

3. 2. コンパイラー

3. 2. 1. PHASE Ⅰ 及び PHASE Ⅱ (ID 及び ED)

PHASE Ⅰ と PHASE Ⅱ の FLOW CHART を第14図に示す。**3.** の初めにも述べたように特に操作もしくは動作が複雑と思われる所以外の操作 box (または operation box) は FLOW CHART より省いてある。各々の操作 box の意味については，次の通りである。

BOX 1 及び BOX 2: ここでは **2. 3. 1.** (2)で述べたような ED table を作る。

BOX 3: **2. 3. 1.** (3)で述べたような FILE-CONTROL に関する ED table を作る。

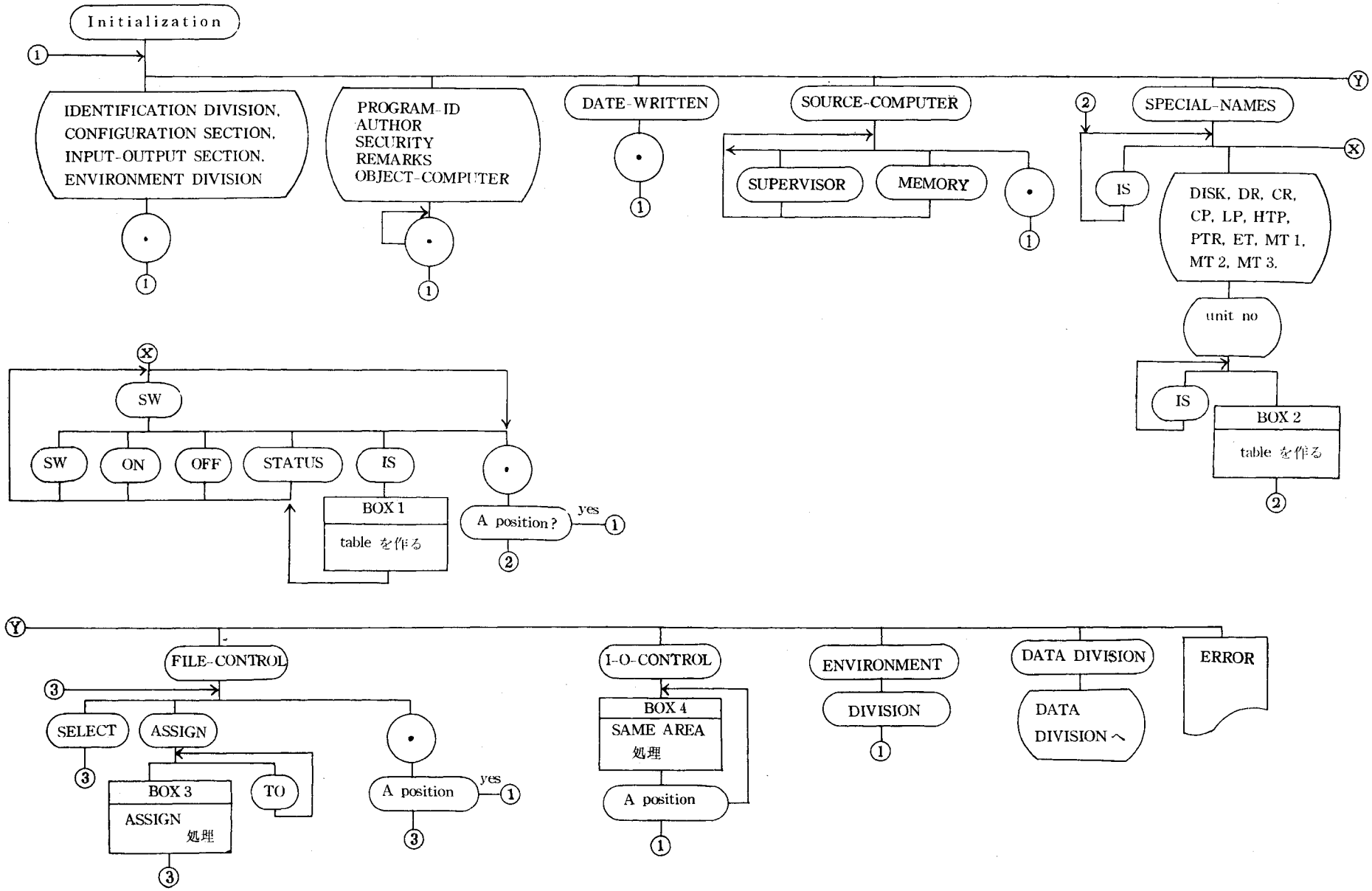
BOX 4: **2. 3. 1.** (4)で述べたように I-O-CONTROL の SAME AREA 処理をなし，ED table 中に同じ AREA が宣言されたものを鎖のように結んでおく。

3. 2. 2. PHASE Ⅲ (DD)

PHASE Ⅲ の大体の FLOW を第15図に示す。

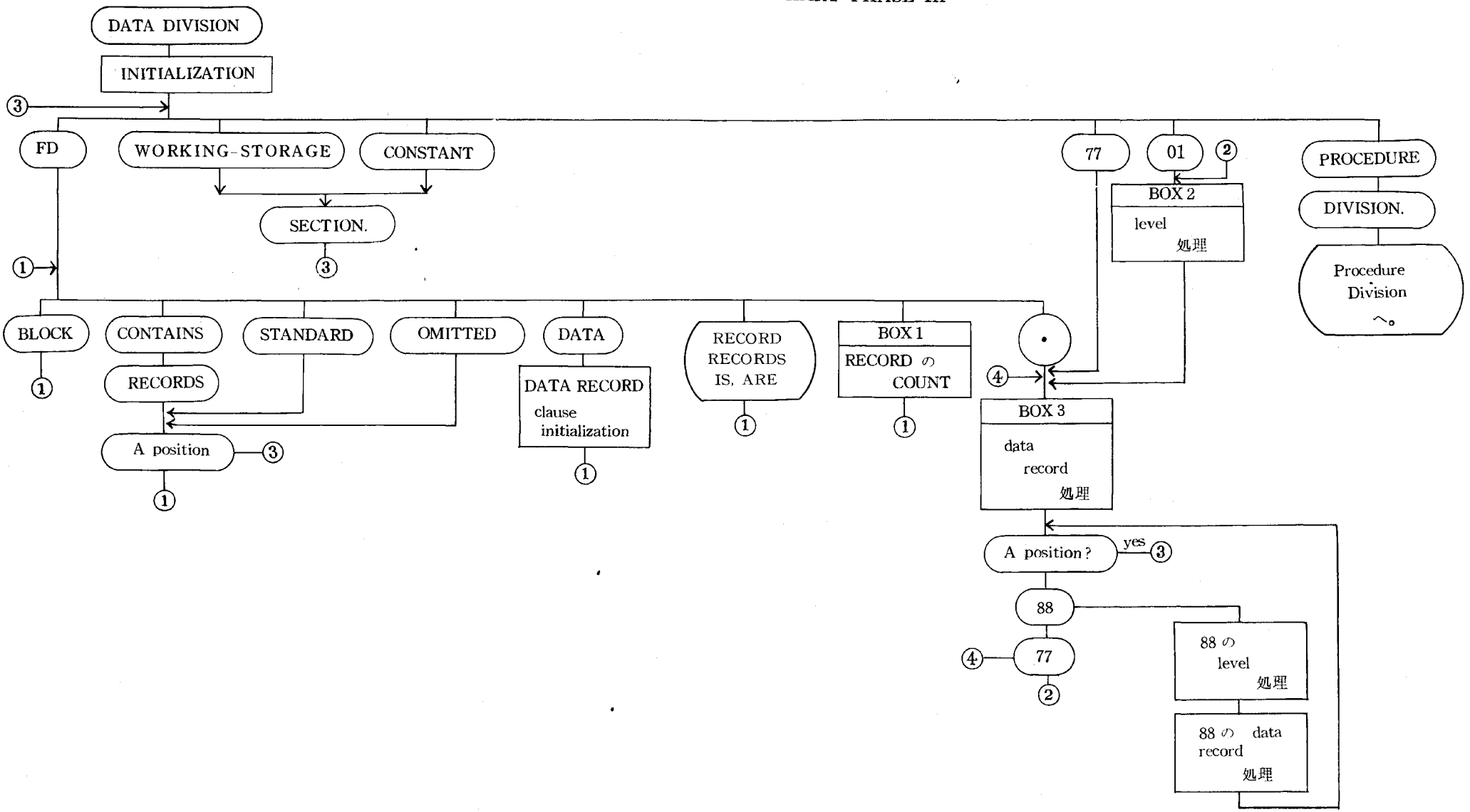
(1) BOX 1: ここでは DATA RECORD clause の処理をする。COBOL-H では FLOW CHART でこの BOX の左に現われている Key Word 以外の語をすべて Data Record の名前として Record の Count に加える。後に BOX 2 内の level 処理ルーチン内で，この Record の数だけ，この

COBOL-H FLOW CHART PHASE I & PHASE II



第 14 図

COBOL-H FLOW CHART PHASE III



第 15 図

File 内のレコードが，同じ先頭番地を持つことになる。

(2) BOX 2: このルーチンは COBOL-H の PHASE Ⅲ では最も重要かつ複雑な動作をする所であって，その完全な叙述をするには多くの努力と根気が必要である。本稿ではその細目を割愛し，大体の機能を述べるに止める。

BOX 2 で行なう操作は，大体次の通りである。

(1) I-O-CONTROL の宣言に基いて，file の先頭番地を定める。

(2) DATA RECORD clause の宣言に基いて file 内の各 Data Record の先頭番地を定める。

(3) level number を読んで，data-name 間の従属関係を知り，割付番地を確定する。

(4) REDEFINES clause に基づいて割付番地を set しなおす。

要するに core メモリへの番地割付を司る部分であるが，COBOL では上述の(1)，(2)，(3)，(4)のような要因で割付番地が，通常の順序と狂うので非常に面倒なコンパイル手続きを踏む必要がある。筆者の場合 COBOL-H の debug で一番時間と努力を費やしたのはこの部分であった。これから COBOL コンパイラーを組まれる方は，Coding の段階以前にこのメモリ割付の部分に特に注意を払うべきであると筆者は考える。

(3) BOX 3: このルーチンも COBOL-H の PHASE Ⅲ の中枢をなす部分で，そのなすことは大体次の事柄である。

(1) FILLER の処理

(2) REDEFINES clause の処理

(3) OCCURS clause の処理

(4) COMPUTATIONAL clause の処理

(5) PICTURE 及び VALUE の処理

要するに BOX 3 では level number 以後書かれている COBOL-H の

source program 全部の処理を行なって，先に述べた DD table, Actual table 等を作成し，かつ初期値，定数等を meso deck として出すことを行なうのである。

以上述べた他に PHASE Ⅲ で良く利用されたルーチンに次のものがある。

(4) data-name-read ルーチン　これはすでに定義された data-name または literal を読んで，その情報を一定の形式にまとめあげる仕事をするルーチンで PHASE Ⅳ でも良く使われるものである。

その情報は下図に示すようにまとめあげられる。

inf–1	x, w, ch, d, dim, n
inf–2	x, w, ch, n

x : =3　non-numeric な literal のとき

=2　numeric literal のとき

=1　computational data-name のとき

=0　以上のいずれでもない場合。(2 bit)

w :　語数を set する。(6 bit)

ch :　文字数を set する。(9 bit)

d :　subscripted variable の場合 inf–2 の n にコンパイル時にわかる address がある場合 1，さもなければ 0 を set する。(1 bit)

dim :　添字付のとき 1、さもなければ 0。(2 bit)

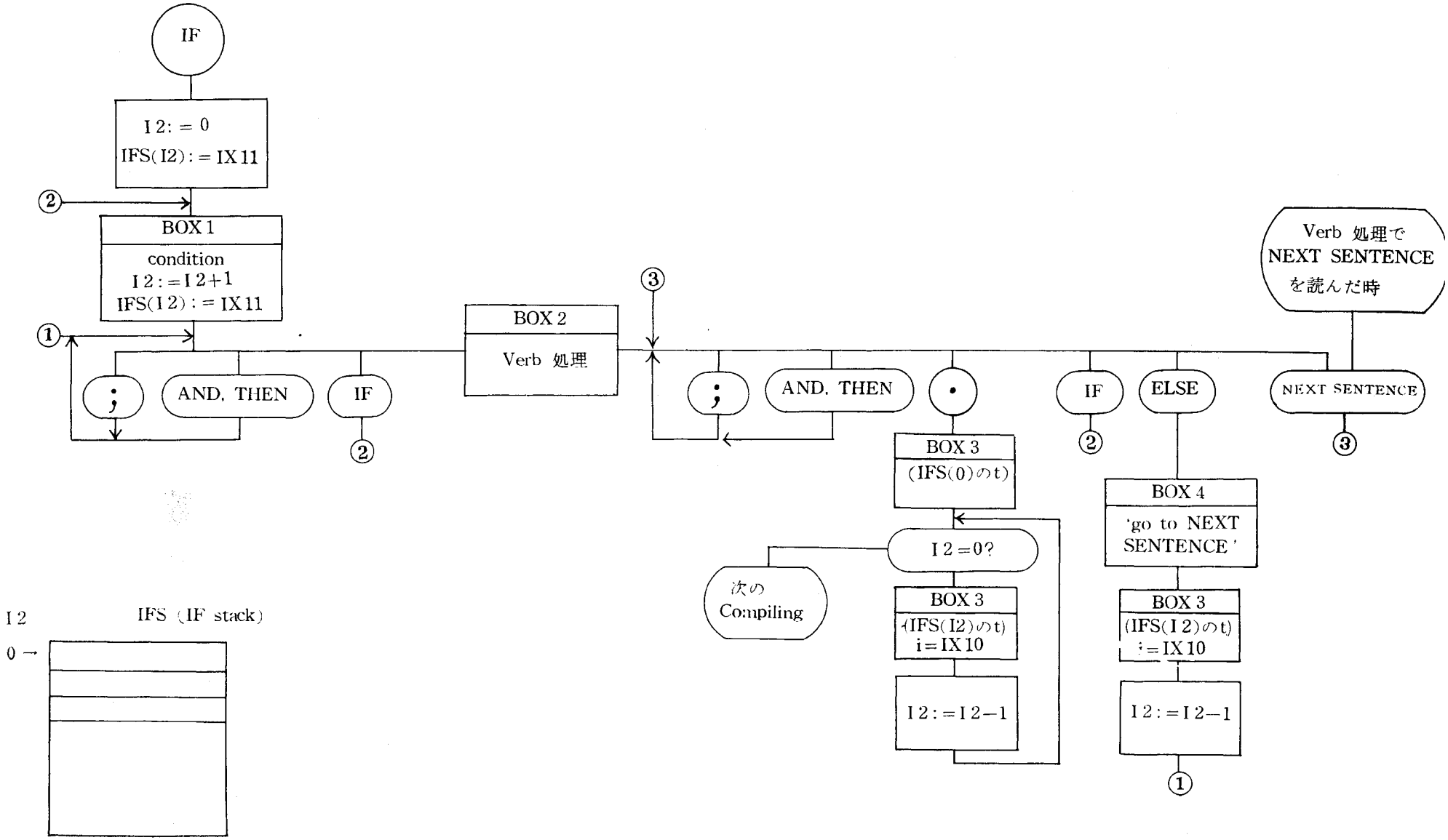
n :　x=3 のとき Actual table address.

n=1 のとき　同 上

以上以外のとき table の location が入る。(14 bit)

inf–1 と inf–2 の bit の意味は同じであるが，inf–2 は添字付 data-name の場合にのみ，意味を持つ。例えば S(I) という data-name がこのルーチンによって処理されると，S に関する情報が inf–1 に I に関する情報は inf–2

COBOL-H FLOW CHART PHASE IV IF clause



第 16 図

に set される。

このルーチンでは name の修飾も処理する。なお COBOL-H では name の修飾は10重まで許す。

3. 2. 3. PHASE Ⅳ (PD)

PHASE Ⅳ の FLOW は各動詞によってそれぞれのコンパイル先に jump させ，各動詞のコンパイルが終った時に一定の番地の内容の示す jump 先に間接的に jump させるようになっている。動詞のコンパイル後の戻り番地を，間接的に指定するようにしたのは，条件文等のコンパイルの際に自由に変更できるのが便利であるからである。

以下，COMPUTE, IF, MOVE の各項について叙述する。他の動詞のコンパイル方法は自明であろう。

(1) COMPUTE　この動詞及びこれに続く formula のコンパイルは若干の部分を除いて通常の数式翻訳と本質的に異なる所はない。筆者の採用した方法は[7]で，筆者が Booleen Expression で試みた方法と類似のものである。

COBOL では数値データとして内部2進式のものと DISPLAY 式のものとがあるが COBOL-H では numeric データはすべて，一旦 DPF に変換して四則演算等を行なうようにコンパイルした。従って DPF 以外のデータ間の演算（COMPUTATIONAL 以外のもの）には必らず演算以外に変換も行なわれる。

(2) IF　IF のコンパイルの方法としては[8]の方法を若干変更して採用している。第16図に IF clause の FLOW CHART を示す。図中 IFS は IF stack という table を示し，I 2 は IFS の pointer である。IFS は I2=0 より用い，IFS (I 2) は IFS 中の I 2 の指し示す table の内容を表わす。

IX 10 は object program のメモリ割付番地の counter で，IX 11 は object program を meso deck の形で，buffer 内に入れておく時の counter

である。

BOX 1： ここでは conditional relation をコンパイルする。COBOL-H の relation test の object program は第17図に示す通りである。

n	i	ind

第 17 図

n： この relation test が成立しない時の object program の jump 先。(14 bit)

i： GREATER THAN, LESS THAN, EQUAL の各 relation test ルーチンの先頭番地。(14 bit)

ind： 上の relation が否定された時に 1 さもなければ 0 を set する。(1 bit)

上で述べたように relation test routine は高々6種しかないので，コンパイラーはこの relation test ルーチンの前に，データの種類によってそれらをそれぞれ標準形に揃えて一定の番地に格納する object program を OUTPUT している。COBOL-H では数値データはすべて DPF の形にして比較し，non-numeric データは，そのままの形で比較している。

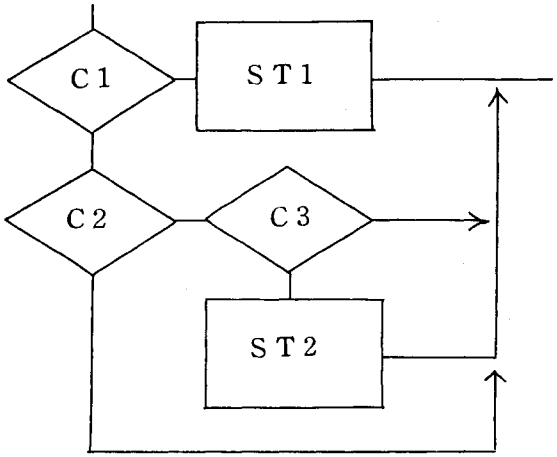
BOX 2： ここでは通常の verb 処理が連続して行なわれるが，;, THEN, ., IF, ELSE, NEXT SENTENCE が現われた時にはこの BOX から出る。

BOX 3： IX 10 (object pregram のメモリ割付 counter) の内容を IFS の指定された番地の内容が示す番地の t part (OKITAC-5090 H では 42～29 bit) に格納する。

BOX 4： ここでは GO TO NEXT SENTENCE に対応する object program を OUTPUT する。実際には「IFS の I 2=0 に対応する内容の示す番地の t-part の示す番地に通訳ルーチンのコントロールを渡す」という命令を出せばよい。

なお，筆者が参考にした［8］の方法によると，この部分に関して若干

IF C1 ST1 ELSE C2 IF C3 NEXT SENTENCE ELSE ST2.



Appeared word	loc.	object program	IFS	I2
IF	p_0	IF	$\bar{p}_0$	0
C1	p_1	C1	$\bar{p}_0$ $\bar{p}_1$	1
ST1		ST1	$\bar{p}_0$ $\bar{p}_1$	1
ELSE		go to (p_0のt)	$\bar{p}_0$	0
	p_2	p_2		
IF			$\bar{p}_0$	0
C2	p_3		$\bar{p}_0$ $\bar{p}_3$	1
IF			$\bar{p}_0$ $\bar{p}_3$	1
C3	p_4	C3	$\bar{p}_0$ $\bar{p}_3$ $\bar{p}_4$	2
NEXT SENTENCE			$\bar{p}_0$ $\bar{p}_3$ $\bar{p}_4$	2
ELSE		go to (p_0のt)	$\bar{p}_0$ $\bar{p}_3$	1
	p_5	p_5		
ST2		ST2	$\bar{p}_0$ $\bar{p}_3$	1
	p_6	p_6		
•		p_6	$\bar{p}_0$	0

第 18 図

object program のムダがあるように思われるが，第16図に示した FLOW CHART ではそれを改良したつもりである。

以上の手続きを例を取って説明する。はじめに第18図の記号その他を説明する。

$p_0, p_1, \ldots, p_6$ は object program を作っていった時のそれぞれのメモリ割付番地を示す。これは第16図で IX 10 によって count されているものに対応する。

$\bar{p}_0, \bar{p}_1, \ldots, \bar{p}_4$ は object program を一旦 meso deck にして MT に吐き出すため，出来上った object program を buffer に一時蓄える必要があるが，これらはその際の location であって，第16図の BOX 1 で使用する IX 11 によって count されるものに対応する。従って $\bar{p}_0, \ldots, \bar{p}_4$ 等はコンパイルの際にだけ意味を有するものであって，object time にはこれらは一切姿を消している。

$\boxed{p_6}\longrightarrow$ のように矢印で示してある部分は，この段階で p_3 の実際の割付番地を知ったコンパイラーが，IFS の助けを借りて，jump 先を object program に書き込む操作を表わしている。この操作は第16図の BOX 3 でなされることに対応する。

第18図内の FLOW CHART は COBOL の定める FLOW CHART ([1] VII–17) で例題に対応するもの，いいかえるとこの FLOW CHART を COBOL 言語で書くと上にあげた sentence になる。

例がどのようにコンパイルされるかは，第16図と第18図とから明らかであろう。

(3) MOVE

MOVE verb のコンパイルの FLOW は次のようになる。

1° data-name–1 を data-name-read ルーチンでよむ。(**3. 2. 2.** (4))

2° data-name–2 を data-name-read ルーチンでよむ。(**3. 2. 2.** (4))

3° data-name–2 (行先の data-name) が COMPUTATIONAL ならば 4°

へ，numeric ならば 5° へ，non-numeric ならば 7° へ，editing (PICTURE に編集機能の文字がある場合) ならば 8° へ，grouped ならば 9° へそれぞれ行く。

4° data-name–1 が数値データである場合には，それを DPF の形で AR に持ち出す object program を OUTPUT し，数値データでない時には，データを全然変形せずに，そのまま AR に持ち出す object program を OUTPUT する。

AR のデータ 2 word をそのまま data-name–2 に store する object program を OUTPUT する。10° へ行く。

5° data-name–1 が数値データである場合には，そのデータを数値標準形にして AR に持ち出す object program を OUTPUT し，数値データでない時には，データを non-numeric 標準形にして AR に持ち出す object program を OUTPUT する。

6° AR のデータを data-name–2 の character 数だけ data-name–2 に store する object program を OUTPUT する。10° へ行く。

7° data-name–1 の種類にかかわりなく，そのデータを non-numeric 標準形にして AR に持ち出す object program を OUTPUT する。6° へ行く。

8° data-name–1 が数値データの場合には，それを数値標準形にして AR に持ち出す object program を OUTPUT し，数値データでない場合にはデータを non-numeric 標準形にして AR に持ち出す object program を OUTPUT する。

AR に数値データがある場合には，それを data-name–2 の PICTURE に従って編集する object program を OUTPUT し，10° へ行く。数値データがない場合には 6° に行く。

9° data-name–2 に word 単位で転送する命令を object program とし

て OUTPUT する。 date-name–1 のデータの長さと data-name–2 のデータの長さとが比較され，data-name–1 の方が長いときには，その分だけが word 単位で切りつめられ，data-name–2 の方が長い時には word 単位で，余った部分に space が埋められる。同じ長さの時にはこのような操作は行なわれない。

10° 次の source program をよみ，(カンマ) がきた時には 2° に行く。それ以外の時には次の compiling を行なう。

以上の説明中に出てきた数値標準形，non-numeric 標準形については **3.3.** の COMPUTE 命令の項で説明する。

3. 3. 通訳ルーチン

2. 3. 4. (3) で述べた各命令について通訳ルーチンの動作を説明する。(**2. 3. 4.** (3) 参照)，簡単なものについては叙述を省略する。

«ALTER»

object time に残された PD table (**2. 3. 4.** (4) END 命令の項参照) によって p_1 の示す object program 上の位置を知り，そこにある GO TO 命令 (DEPENDING option のないもの) の行先を p_2 と置きかえる。(注, ALTER 命令の procedure-name–1 は GO TO statement のみよりなる命令でなくてはならない)

«CLOSE»

f より Actual table 中に OPEN bit があるかどうかを調べ，ないときにはエラーとして Monitor にコントロールを移す。あるときには OPEN bit を Reset して EOF (end of file) を書き，(LABEL RECORD が OMITTED の場合には TM (Tape Mark) よりなる EOF をかく) 磁気テープを指定により巻き戻したり，本体から切り離したりする。

«COMPUTE, MOVE»

すでに **2. 3. 4.** で述べたように COMPUTE verb, MOVE verb 及びそれ

に続く数式は**2. 3. 4.** にあげた個々の命令に分解される。それらの機能については**2. 3. 4.** で述べられているので，ここでは数値標準形と non-numeric 標準形とについて述べる。

○数値標準形（第19図参照）

数値標準形では，数値は 4 word で表現される。上 2 word には数値の整数部分が character モードで右につまった形で入り，下 2 word に小数部分を左につめた形で入れる。余白は0とする。

数値	d_{14}	d_{13}	d_{12}	d_{11}	d_{10}	d_9	d_8
	d_7	d_6	d_5	d_4	d_3	d_2	d_1
	f_1	f_2	f_3	f_4	f_5	f_6	f_7
	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}
符号							

数値標準形

第 19 図

例えば 123.56 は $d_3=1$, $d_2=2$, $d_8=3$, $f_1=5$, $f_2=6$ で他はすべて0となる。なお，符号は別に 1 word を用意して蓄えるようになっている。

○non-numeric 標準形（第20図参照）

この標準形は 2 word 14 桁である。14 桁未満の data をこの標準形にすると，データは左側につめられ余白はブランクがつめられる。

c_1	c_2	c_3	c_4	c_5	c_6	c_7
c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}

non-numeric 標準形

第 20 図

また，この標準形を n 桁格納するときには，左より n 桁だけがしまわれる。

例えば PICTURE が 9(5) で内容が 00123 というデータをこの標準形にすると

00123⊿⊿⊿⊿⊿⊿⊿⊿⊿⊿

となる。また，これを PICTURE として X(6) をもつ data-name に格納したとすると

00123⊿

というようになる。COBOL-H では MOVE 命令（第12図参照）の際に

type の異なる elementary item 間のデータの転送には原則としてこの標準化の手順が踏まれる。

«OPEN»

file-name に対応して Actual table 上に確保された table 中に，この file が OPEN されたことを登録する。

CARD に ASSIGN された file が INPUT として OPEN されると，通訳ルーチンは，この file に割当てられた buffer area にあらかじめカードを読みこむ命令を出す。

MT に ASSIGN された file の場合には，INPUT の場合に label のある時は file の名前を持つ BOF をさがして，第1レコードを読みとる位置に MT のヘッドを set し label のない時には一つだけ TM (Tape Mark) を読む命令を出す。

OUTPUT の場合，label record がある時はそれを書き，ない場合は TM のみを書く。

LP に ASSIGN された file に対しては別段なにもしない。

«PERFORM»

p_2 より procedure-name–2 の範囲（THRU のない時には procedure-name–1 の範囲）を知って，その終りの object program を一旦 i_0 以下に save し，そこに i_1 に jump する命令を植えつける。（このために COBOL–H では各 procedure-name が現われるたびに non effect の object program を 1 word づつ output している。）

この後にコントロールを object program の p_1 で指された部分に渡す。

p_1 から p_2 迄 Execute し終ると，コントロールは先程植えておいた命令により i_1 に戻るが，ここにはさらに先程 save した p_2 の終りの data を再び元に戻す命令があり，それを実行した後，元の流れ（現在 execute した PERFORM verb の次）に戻る。

«READ»

指定された file の 1 record を読みこむ。もし file の終りが検知された場合には，AT END 以下の命令にコントロールを渡す。AT END clause がない場合には，次の命令にコントロールを渡す。

«STOP»

STOP RUN clause が execute されると LP を 1 page 改頁し，Monitor に control を渡す。

«WRITE»

file の情報により，指定された入出力機器に data を打ち出す。

«FEED»

LP を改頁する。なお，これは COBOL-H 固有の命令である。

3. 4. モニター

モニターは常時 core にあって MT の入出力エラー等を監視し，かつエラー処理を行なう他，COBOL-H プログラムの連続処理を容易にするモニターカードの処理，system をロードする働きがある。

3. 4. 1. MT のエラー処理

MT の入出力でエラーが生ずると割り込みがかかり，もし WRITE 命令の error ならば，1 record MT を巻き戻し erase を行なった後に再び同じ record を書く。入力の場合には 1 record だけ巻きもどした状態で machine stop する。restart で，同じレコードを読み直す。

3. 4. 2. Monitor Card の処理

OKITAC-5090 H のコンソールにある TOTAL のボタンを押した後 SCC 455 (8 進) より start させると Monitor は数字で表わされた命令に従って次のような働きをする。(数字で表わされた命令については [2] 参照。)

1° CARD を 1 枚，WS 以下によみこむ。(WS は Working Storage (作

業用エリア）の略。）

2° WS 以下によんだものが MONITOR CARD でなければ 1° に行く。

3° WS 以下の内容を AR に持ってくる。system の initialization を行なう。

4° AR を左に 1 桁シフトし，よみとった数字に対応させて定められたルーチンに jump させる。(jump 先でそれぞれの命令を行なって，1° または 3° または 4° に戻る。3° に戻るときには，jump 先でシフトされた AR を WS 以下に格納しておく。4° に戻るときは AR の内容がこわされていない時である。)

各命令後の戻り場所は次の通りである。

1° に戻るのは命令 6 の場合

3° に戻るのは命令 1，2，3，4，5，7 の場合

4° に戻るのは命令 0，8，9 の場合

である。

3. 4. 3. system ローダー

COBOL-H では system は 1) MONITOR, 2) 通訳ルーチン, 3) コンパイラー（プリコンパイラーを含む）の三つの部分にわかれている。

モニターには 2), 3) のローダーが命令 1，4 に対応してあるが，さらに manual によって簡単に system の Read と Write とが行なえるようになっている。この機能は新しい system を MT に書きこむ時に使用される。

なお，モニター自身はブートストラップでロードできるが，一度入ったブートストラップはその後もこわされずに core に残っている。

参 考 文 献

[1] Revised Specifications for a COMMON BUSINESS ORIENTED LANGUAGE (COBOL) for Programming Electronic Digital Computers, Department of Defence, 1961.

[2] 穂鷹良介 COBOL-H 説明書 1967.
[3] 日本電子工業振興協会 COBOL (OKITAC-5090H 用) 昭和42年1月。
[4] BEMA/DPG X 3, 4 COBOL INFORMATION BULLETIN #5 November 17, 1964.
[5] OKITAC-5090H MASH 沖電気工業株式会社データ処理営業部 昭和39年4月1日。
[6] D.D. McCracken A guide to COBOL programming, John Wiley & Sons, Inc., New York, 1963.
[7] 穂鷹良介 「Automatic Coding について (Ⅱ)」 —— Boolean Expression の翻訳 —— 商学討究 第15巻第2号。
[8] 日本電子工業振興協会 「COBOL Compiler の実験報告書」 1964年3月。

＜略　　称＞

P.C.S.P.	Pre-compiled Source Program	2. 2.
ID	Identification Division	2. 3.
ED	Environment Division	2. 3.
DD	Data Division	2. 3.
PD	Procedure Division	2. 3.
LSC	Least Significant Character	2. 3. 2.
SW	Switch	2. 3. 2.
MT	Magnetic Tape	2. 3. 2.
PTR	Photo Tape Reader	2. 3. 2.
ET	Electronic Typewriter	2. 3. 2.
LP	Line Printer	2. 3. 2.
HTP	High Speed Tape Puncher	2. 3. 2.
FD	File Description	2. 3. 3.
DPF	double precision floating type data	2. 3. 4.
AR	Arithmetic Register	2. 3. 4.
TM	Tape Mark	3. 3.
WS	Working Storage	3. 4. 2.