コンピュータ・ネットワークにおける，リスト処理・ポーランド記法とアーキテクチャについて

渋井　二三男

本稿ではコンピュータとNetworkの中で重要かつ難攻不落の“言語”，“リスト処理・ポーランド記法”と“アーキテクチャ”の関係について中型コンピュータモデルをSampleとして取り上げ詳細に論述する。

1. 電子計算機の構造とアセンブラ

本稿では，最初に計算機の構造，機械語及びアセンブラ言語の関係についてのべる。ここでは説明の具体例として，中型コンピュータモデルを例に論述する。

2.1　一般的な計算機構造

計算機は一般的に命令翻訳部，ロケーション・カウンタ，命令レジスタ及び各種の作業レジスタと汎用レジスタによって構成されている。このほかに計算機と記憶装置の間のインタフェースは記憶番地レジスタと記憶バッファ・レジスタがある。

図1.1は中型コンピュータのシステムのハードウエア構成図である。記憶装置は情報が格納される場所である。プロセッサはこの情報に関する内容処理が行われる。いずれも１と０の形で格納されている。記憶されている場所はアドレスによって指定され，各アドレスは特定のバイト・ワード又は文字を識別する。

計算機の基本動作は，コアメモリの中に格納されている命令を順次１命令ずつ実行していくことである（命令は機械しか理解出来ない言語を機械語という）。命令の種類や機能，長さ，実行速度などは，計算機の各モデルによってことなる。使用されているワードは，データワード（Data-Word）と命令ワード（Instraction-Word）がある。前者は計算機内部の数値単位であり，後者はプログラムの内蔵方式で計算機を動作させている。計算機の命令はその種類を機能別に分類すると，演算命令，分岐命令，制御命令と入出力命令がある。

演算命令の一般的な実行形式はＡ番地の内容とＢ番地の内容を演算して，その結果をＣ番地に格納される。レジスタをもたない計算機はコアメモリＡ番地にある内容とＢ番地にある内容を演算して，その結果をＣ番地に格納する。又はＡとＢとが無関係なＰ番地に格納してもよい。こ

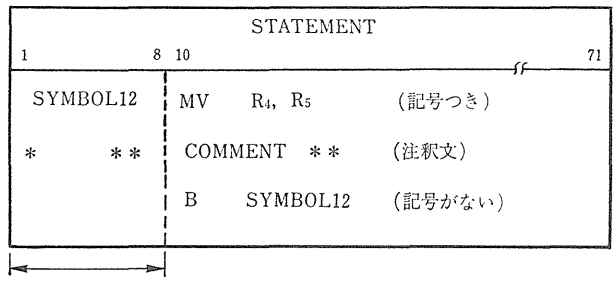
れらはいずれも計算機の設計方式によってことなる（図1.2は演算命令の実行過程である）。

1.2 ステートメントの型式

言語で書かれたプログラムはソース・プログラムと呼ばれる。このプログラムは，ステートメントと呼ばれる単位から構成されている。ステートメントには，ラベル，オペレーション，オペランドおよびコメントの4つのフィールドから構成される。以下，各フィールドについて説明する。

（1）ラベル・フィールド（Label Field）

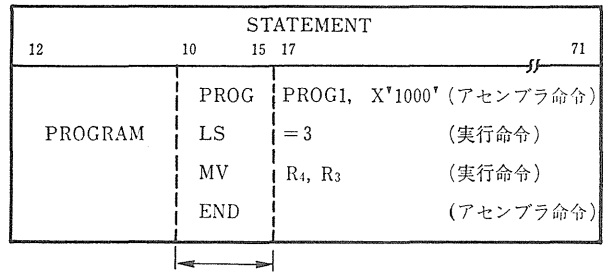
このフィールドは，プログラマが必要に応じて定義する部分で，ステートメントを識別するシンボルを記入する場所である。このフィールドに記入する記号は8文字以内で，最初の文字は，1桁目から記入されなければならない。1桁目がブランクの場合は，アセンブラは名前の記号がないものとみなす。又1桁目がアステリスク（＊）の場合は，コメントとみなす。ラベル･フィー



ルドの記入例を次に示す。

（2）オペレーション・フィールド（Operation Field）

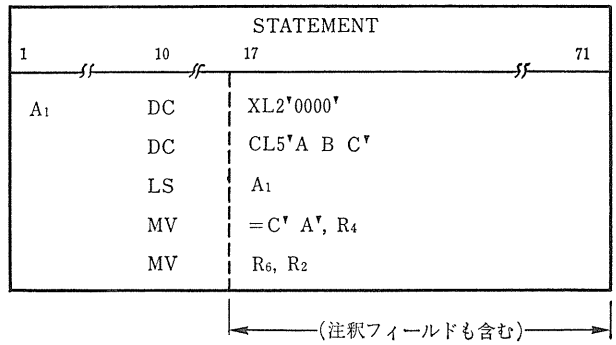
このフィールドは，そのステートメントの機能を表わす名前である。その機能は実行命令，機械命令，マクロ命令，指示･宣言など4種類に分類することが出来る。命令コードは10桁目から



記入する。フィールドへ記入出来る文字数は6文字である。その記入例を次に示す。

（3）オペランド・フィールド（Operand Field）

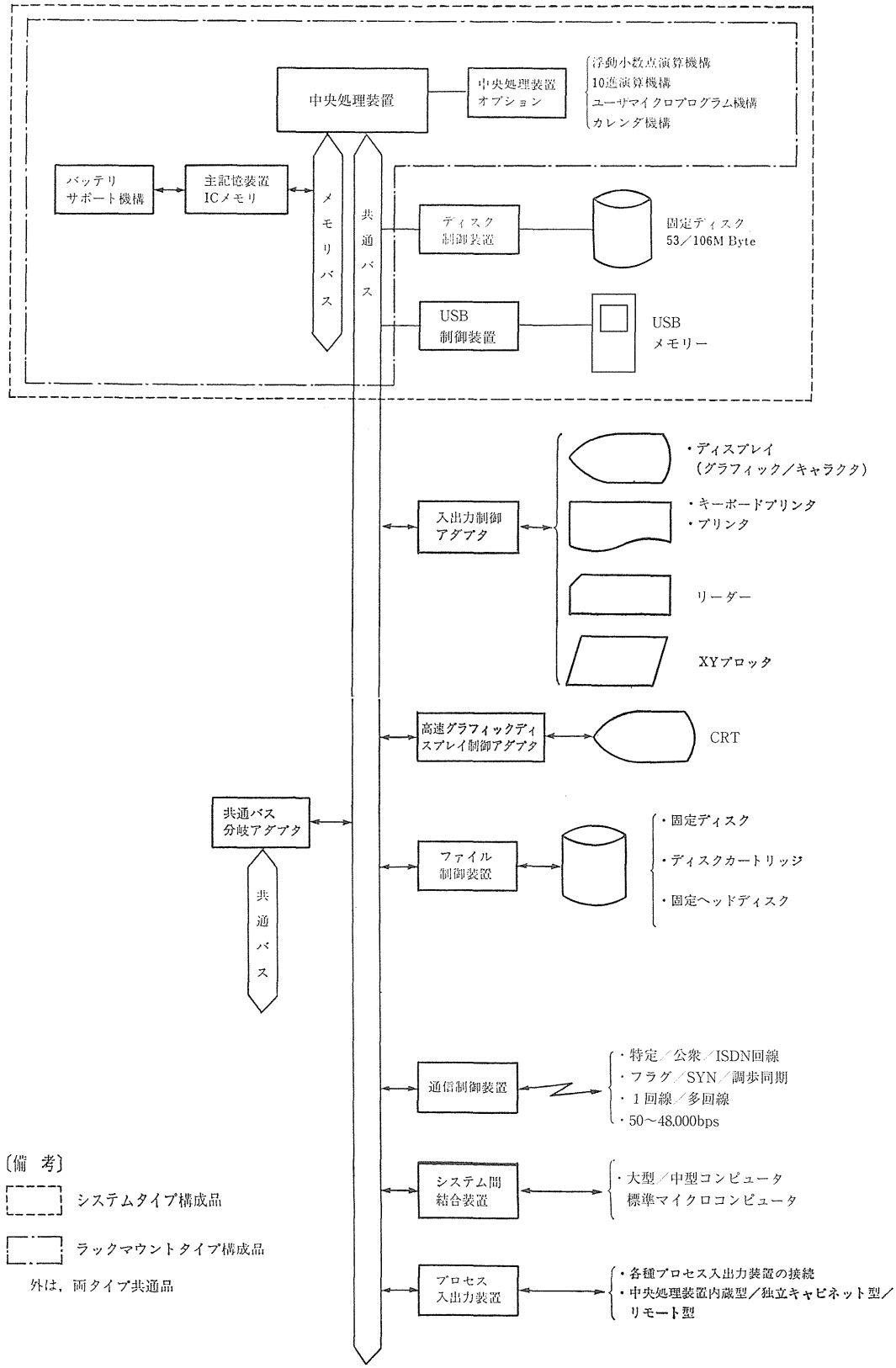
このフィールドは，ステートメントの機能に対するオペレーションコードの補充説明である。この命令にはオペランド部を必要としない命令と1個以上のオペランドからなるオペランド部を必要とする命令に分けることが出来る。このフィールドに記入される事項は，その命令フィールドに記入されている命令が行うオペレーションの対象となるデータを表現したり説明したりするためのものである。例えば，記憶位置，レジスタ，記憶域の長さ，あるいはデータの種類などを表す情報である。

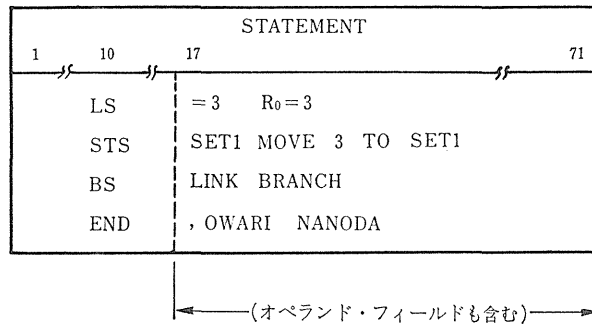


（4）コメント

コメントは，プログラムに関する説明・注釈などを記述するためのものである。プログラム・リストに一緒に印刷される。コメントでは，ブランクをも含めて，256字のすべての有効な文字を使うことが出来る。その記入方式を次のように示す。

図1.1　沖電気製中型コンピュータのシステム構想概念図





2.3　情報の表現形式

計算機において取り扱われる情報は文字と数値である。即ち，

英字と標準記号　　A～Z及び¥＃@

数　字　　0～9

特殊文字　　＋－，＝・（　）▼／＆及びブランク

対になった引用符（▼）の間，コメントの中など文字を用いることが出来る場所であればどこでも，256種数のうち任意の文字を指定することができる。図1.2及び図1.3はEBCDICコード表である。

中型コンピュータのモデルにおいて，主記憶装置内のデータは，8ビットからなるバイト単位で表示される。中央処理装置又は入出力制御装置と主記憶装置間の転送は，バイトの整数倍で行われる。バイトがいくつか連続した集まりをフィールドという。フィールドの長さが命令によってあらかじめ定まっているのを固定長，命令により指示するのを可変長という。

固定長のフィールドの長さは1，2，4，8バイトのいずれかである。2バイトの長さのフィールドはワード，4バイトのフィールドをダブルワードという。各フィールドの各ビットには0から始まる一連番号を左から右へとつける（図1.4，図1.5，図1.6を参照）

2.4　命令の形式とその機能

計算機の制御方式には，1－アドレス，2－アドレス，又は3－アドレス方式に分類することが出来る（図1.7を参照）。

（1）1－アドレス方式

この方式はあらかじめ約束されたアキュムレータ(ACC)がある。アドレスAで指定されたデー

図1.2　EECDIC（カナ入り）コード表

B_5~B_8 ＼ B_1~B_4	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	NUL		DS		SP	&	−			ソ						0
0001			SOS				/		aア	jタ			A	J		1
0010			FS						bイ	kチ	sヘ		B	K	S	2
0011			TM		⌞				cウ	lツ	tホ		C	L	T	3
0100	PF	RES	BYP	PN					dエ	mテ	uマ		D	M	U	4
0101	HT	NL	LF	RS					eオ	nト	vミ		E	N	V	5
0110	LC	BS	EOB	UC					fカ	oナ	wム		F	O	W	6
0111	DL	IL	PRE	EOT					gキ	pニ	xメ		G	P	X	7
1000									hク	qヌ	yモ		H	Q	Y	8
1001									iケ	rネ	zヤ		I	R	Z	9
1010		OC	SM		C	!		:	コ	ノ	ユ	レ				
1011					・	¥	,	#				ロ				
1100					<	*	%	@	サ		ヨ	ワ				
1101					(	)	−	▼	シ	ハ	ラ	ン				
1110					+	;	>	=	ス	ヒ	リ	〃				
1111					|	¬	?		セ	フ	ル	。				

制御符号

NUL:Null
PF :Punch Off
HT :Horizontal Tab
LC :Lower Case
DL :Delete
TM :Tape Mark
RES:Restore
NL :New Line
BS :Black Space
IL :Idle
CC :Cursor Control
DS :Digit Select

SOS:Start Significance
FS :Field Separator
BYP:Bypass
LF :Line Feed
EOB:End Of Block
PRE:Prefix
PN :Punch On
RS :Reader Stop
UC :Upper Cace
EOT:End Of Transmission
SM :Set Mode
SP :Space または Blank

▨ 文字セットで指定されたコード

英字と標準記号
A～Z¥＃@

数　字　0～9

特殊文字
＋－，＝・＊（　）▼
／&ブランク(SP)

¥は¥または$

タとこの *ACC* との間で演算を行い，その結果を再び *ACC* へストアするようになっている（例えば，固定長）。

(*ACC*) ＊ (*A*) → *ACC*

図1.3　定数の形式を表すコード

コード	定数の形式	機械内での形式
C	文字	1文字につき8ビット・コード。　Ln＝1～256 境界への整列は行なわれない。
X	16進数	16進数1桁につき4ビット・コード。Ln＝1～256 境界への整列は行なわれない。
F	固定小数点数	符号付または符号なしの固定小数点2進数形式。 Ln＝1～4 長さ指定を省略したときは2バイトで，2バイト境界への整列が行なわれる。
E	浮動小数点数 UMOS/Cでは使用不可	短精度の浮動小数点数形式。Ln＝なし 必ず4バイトで，4バイト境界への整列が行なわれる。
D	浮動小数点数 UMOS/Cでは使用不可	長精度の浮動小数点数形式。Ln＝なし 必ず8バイトで，8バイト境界への整列が行なわれる。
A	アドレス	アドレスの値　Ln＝1～2 長さ指定を省略したときには2バイトで，2バイト境界への整列が行なわれる。

図1.4　フィールド

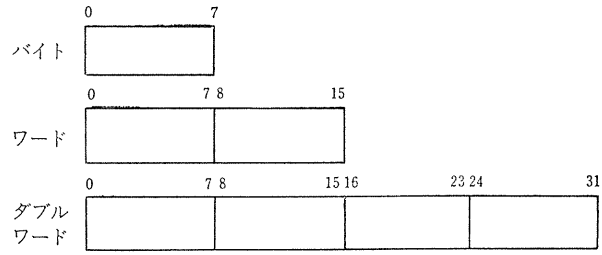


図1.5　情報の形式

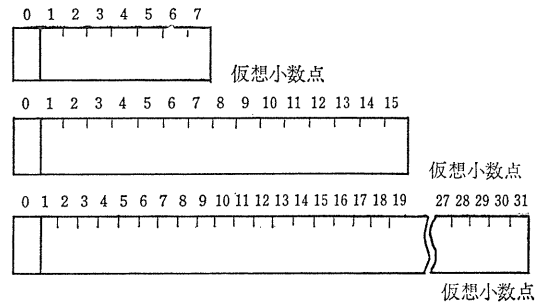


図1.6　論理演算のデータ

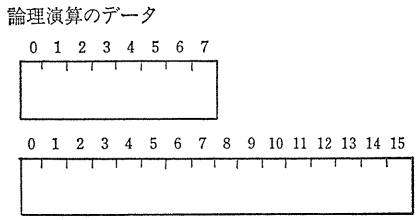
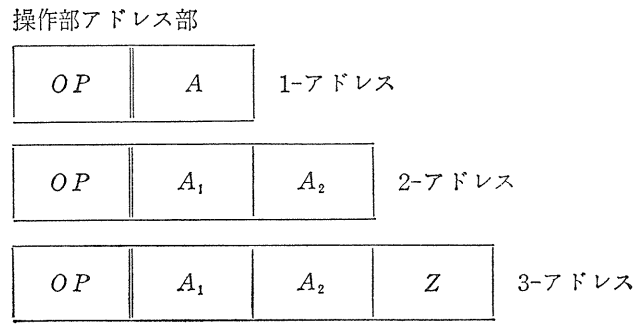


図1.7　1－アドレス，2－アドレス，3－アドレスの命令形式



（2）2－アドレス方式

2－アドレス方式の命令はアドレスA_1の内容とアドレスA_2の内容との間で演算が行われ，その結果をアドレスA_1にストアするようになっている（例えば，可変長)。

(A_1) ＊ $(A_2) \rightarrow A_1$

（3）3－アドレス方式

3－アドレスはアドレスA_1の内容とアドレスA_2の内容との間に演算が行われる。その結果をアドレスZにストアするように構成されている。

(A_1) ＊ $(A_2) \rightarrow Z$

一般にアドレスには，論理アドレスと物理アドレスがあり，前者はプログラムで指定するアドレスをいう。後者は実際の主記憶装置のアドレスをいう。中型コンピュータモデルでは命令で指定される論理アドレスには，16ビットであるが，主記憶装置をアクセスするときに，4ビットの拡張アドレスが付加されて20ビットになる。4ビットの拡張アドレスは，セグメントブロック番号を指定し，連続した16ビットの論理アドレスは，上位5ビットが論理セグメント番号を，下位11ビットがセグメント内変位を指定します（図1.8を参照)。又，図1.9は汎用レジスタと拡張レジスタである。

一般に電子計算機組織の中には，アドレスには主記憶装置の中にあり，レジスタは，演算装置

図1.8　物理アドレス形成の概念

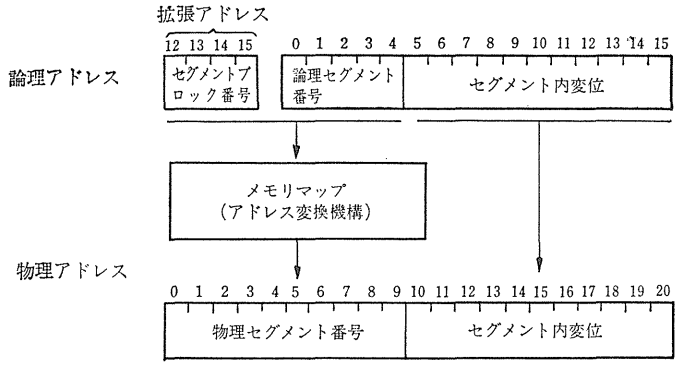
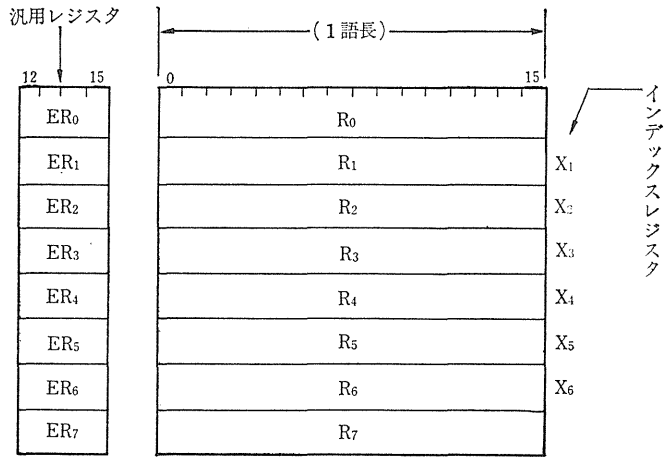


図1.9　汎用レジスタと拡張レジスタ

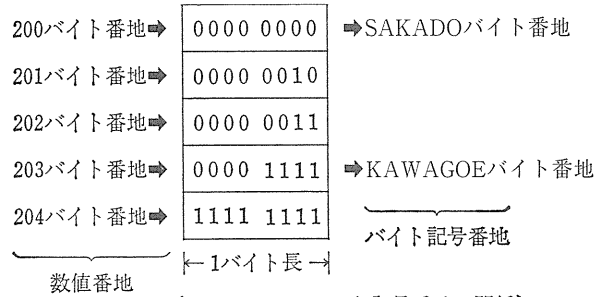


R0～R7　：General Register　0～7

ER0～ER7：Extended Register 0～7

の中に存在する。しかし，プログラムを組む時に用いられるブロック構造図については，主記憶装置と演算装置とを分離して用いる場合と併合して用いる場合とがある。そうして，主記憶装置の中に，実存しながらカウンタなどのレジスタ的な役割を果すのがメモリレジスタである。特に電子計算組織では，数値番地の代わりにワード記号番地を使って表現することが出来る。下記はその使用例で，200バイト番地の代りにワード記号番地「SAKADO」というバイト番地を使用したものである。

実行命令のオペランドでレジスタを指定する場合レジスタ記号を使う。レジスタ記号は，ハードウエアがもつ汎用レジスタ，フローティング・レジスタに対して特別に割り当てた記号を用いて表現する。レジスタ番号の前に英字R，X，Eをつけて汎用レジスタ，インデックス・レジスタ，フローティング・レジスタを表現する。レジスタ記号は全部で16個ある。



〔バイト番地とワード記号番地の関係〕

① 汎用レジスタ記号

ER_0, ER_1, ER_2, ER_3, ER_4, ER_5, ER_6, ER_7,

② インデックス・レジスタ記号

X_1, X_2, X_3, X_4, X_5, X_6,

③ フローティング・レジスタ

F_0, F_2

④ 命令カウンタ記号

I　C

1.5 命令コードの型式

命令の構成は，各電子計算組織によってことなるが，中央処理装置が種々の動作を行なうためには，動作の基本となる指示が必要である。このように，動作の基本となる指示を命令（Instructions）という。

命令の構成は，オペレーション・コード部とアドレス部に分けられ，ワードを形成している。特にアドレス部は各電子計算機によってことなる。一般にアドレス部が1つのものを1－アドレス方式といい，2つのものを2－アドレス方式という。

(1) 基本型

このタイプは，とくに中型コンピュータにおいて一番多く使われている。命令形式がまったく基本のままのシステムである。

OP	A

→1－アドレス方式（“～を～せよ”）

(2) 拡張型

このタイプは中型・大型計算機に使用される。

OP	A_1	A_2

→2－アドレス方式（“～と～を～せよ”）

例えば　A，R_4，M

これはコアメモリM番地の内容をレジスタR_4に加算せよという意味である。

又，A　M，N

この場合はコアメモリM番地の内容をコアメモリN番地の内容へ加算し，その結果がN番地に残る(2－アドレス方式)という表現である。命令はアドレス指定の方式が最も大切である。制限されたコアメモリを有効に使うために種々の方法が考えられて来た。相対アドレス方式，間接アドレス方式，インデックス・レジスタ方式，ベース・レジスタ方式，ページ・メモリ方式等がその典型である。

1.6　基本命令の形式

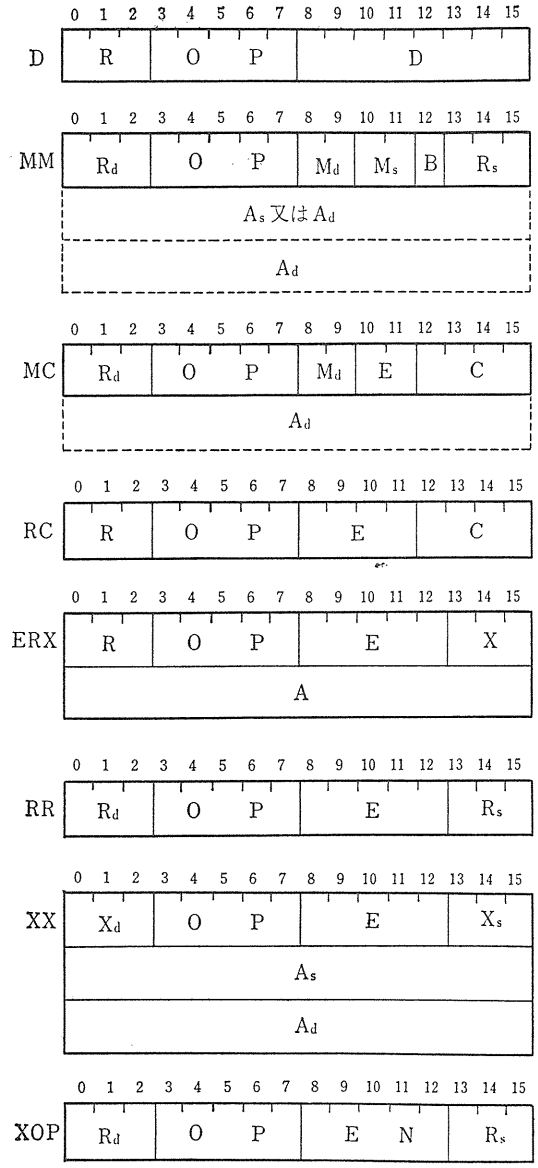
命令の長さは，ワードを単位として1ワード，2ワード，3ワードの3種類がある。主記憶装置上ワード境界におかなければならない（ここでは中型コンピュータモデルを用いて説明する)。

命令は図1.10に示すようにD形，MM形，MC形，RC形，ERX形，RR形，XX形，XOP形の8つの基本形式のうちいずれかに分類される。図1.10モデル・コンピュータ命令の一覧表の一部である。

＜アセンブラ命令＞

実行命令は，プログラムの実行の段階で一連のオペレーションを実行するように計算機に命令するために使う命令であるのに対して，アセンブラ命令は，アセンブリの段階で，ある種のオペレーションを実行するようにアセンブラに指示するために使う命令である。アセンブラ命令のステートメントは，実行命令のステートメントとことなり，かならずしもアセンブルされたオブジェクト・プログラムに機械語を組入れることはない。アセンブラ命令のステートメントには，命令を作り出さず，その代りに定数などのデータを入れるための記憶域を確保するはたらきをもつもの（DC，DSなど）や，アセンブリの段階のみ働きをし，アセンブルされたプログラムの中には何も作り出さず，またアドレス割当カウンタにも何ら影響を与えないもの(EQU)がある。下記は中型コンピュータモデルに使用されている命令の一覧表の一部である。

図1.10　命令形式



R：はん用レジスタ指定，分岐条件指定，浮動小数点演算用レジスタ指定及びオペランドが直接数値であることの指定などに使用する。
R_d，R_sはそれぞれディスティネーション側，ソース側の領域であることを示す。ディスティネーション側とは命令の受け側を意味し，ソース側とは送り側を意味する。

OP：命令コード指定に使用する。

D：相対アドレスの数値及び直接数値などに使用する。

E：OP部を拡張するのに使用する。

M：アドレッシングの指定に使用する。M_d，M_sはそれぞれディスティネ

ーション側，ソース側の領域であることを示す。

B：取り扱うデータの長さを指定する。

B＝0のときはワードデータ

B＝1のときはバイトデータ

X：インデックスレジスタとしてのはん用レジスタ指定に使用する。

X_d，X_sはそれぞれディスティネーション側，ソース側の領域である

ことを示す。

C：直接数値，シフトビット数及びビット位置指定などに使用する。

A：インデックス修飾のメモリアドレスとして用いる。

A_d，A_sはそれぞれディスティネーション側，ソース側の領域である

ことを示す。

EN：制御を司さどるマイクロプログラムを指定するエントリ番号である。

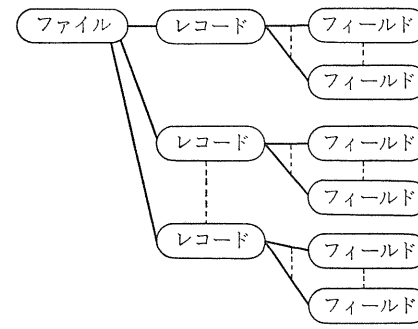
2. リスト処理とアーキテクチャ

次に，本稿では特殊なデータ構造として，リスト（List），トリー（Tree）とリスト処理のアーキテクチャの概念についてのべる。

2.1 情報の論理的構造

コンピュータのデータ構造は，ビット，数字，または文字である。データ構成単位には，ファイル（file），レコード（record），及びフィールド（field）がある。図2.1はその一般的な構造である。

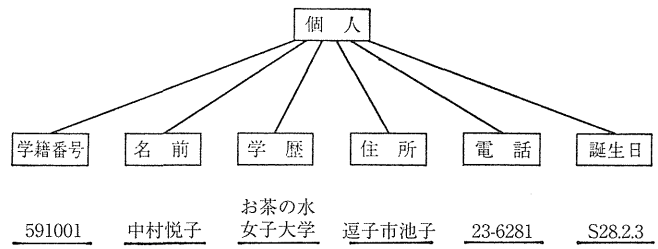
図2.1 データの構造



例えば次のデータについて考えてみる。

最初の591001は学籍番号を示すデータ，盛岡太郎は氏名を示すデータ，皆それぞれことなる種類のデータである。いずれもデータのワク組みという役割をもっている。このワク組み自身には実体は何もない。これをアイテム（item）という。これに対して，23-6281のような実体をアイテムの実現値という。上記の「個人」は6つのアイテムの集まりである。「学籍番号」などのアイテムと区別して，この集まりをグループアイテムとよぶ。グループアイテムにはレベルを考えることが出来る。その一番上のレベルのグループの実現値をレコード（Record）とよぶ。レコードは計算機の処理する1つのデータの単位である。それに対して，アイテムの実現値はそれ以上区切ることの出来ないデータの最小単位と考えることが出来る。1つのアイテムしかないときにはそのアイテムの実現値がレコードになる。

レコードが複数個集まったものを，ファイル（file）という。上記の学生名簿は1つのファイル



であり，ワク組みはその実現値が記憶装置内のどこに格納されているかといったこととはまったく独立した使用者側からみたデータの形式をのべたものであるが，このワク組みと外側から見たその実現値の並び方とを含めて外部構造（External Structure）という。それに対して，その実現値を記憶領域に格納するときの構造を内部構造（Internal Structure）という。一方，データの構造の数学的表現は次のように考えることが出来る。即ち，レコードは1つのベクトル，フィールドはそのベクトルの構成要素，ファイルは，レコード・ベクトルを集めたマトリクスである。

〔フィールド〕

$$F=\begin{pmatrix}R_1\\R_2\\R_3\\\vdots\end{pmatrix}=\begin{pmatrix}f_{11} & f_{12} & \cdots\cdots\\f_{21} & f_{22} & \cdots\cdots\\f_{31} & f_{32} & \cdots\cdots\\\cdots\cdots & & \\\cdots\cdots & & \end{pmatrix}$$

ファイル　レコード　フィールド

フィールドは，データの論理的構造における最小単位である。いくつかの文字又はビットから構成されている。フィールドには，数値フィールド，英数字フィールド，2進フィールドなどがある。

〔レコード〕

レコードは，1つの対象について記述したフィールドの集合であり，データの論理的構造の第2の単位である。又，1つのファイルを構成するレコードには，(1)データ・レコード，(2)ラベル・レコード，(3)索引レコードなどの3種類がある。

データ・レコードは，個々の対象データを記述したレコードであり，ファイルの本体である。

〔ファイル〕

ファイルは，1つのプログラムによって処理されるデータの最も大きな論理的単位である。

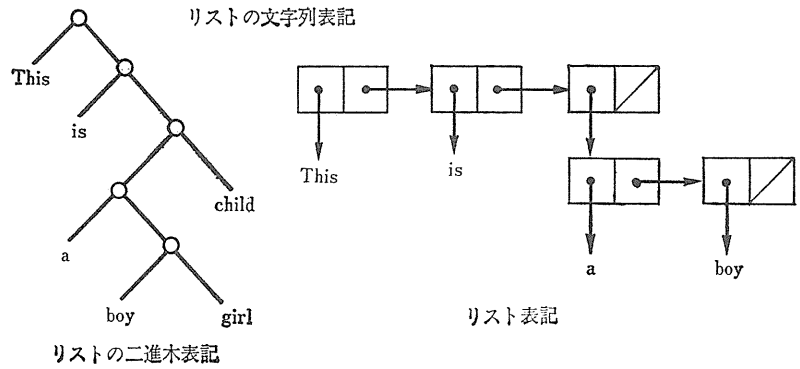
2.2　リスト構造・リスト処理

（1）リストの表記

一般的なデータとしてのリストは，リスト要素を「ポインタ」によって連結したものをいう。すなわち，リストの開始，終了を「(」と「)」で表わし，図2.2はリストの表記である。ここでは文字の表記は計算機の入出力に使われ，二進木表記はリストを概念的な表現に使い，そして箱表記は（各箱にポインタをつけて）計算機内でのデータのあり様を表現するのに使用される。

リスト処理には，リストの要素を取り出し，生成，削除，追加，連結，分割，合併，複写，走

図2.2　リストの表記



査などがある。リスト処理としてとくに注意することは，ポインタの取り扱いとリストの生成消去である。

リストの生成消去はリスト要素をどのように生成するか，不必要な要素をどのように削除したり回収するかという問題である。リストは概念的にひとまとまりをしていて，計算機の中では各要素が分散しているので，その生成消去は記憶空間という問題を含みますので工夫を要する。

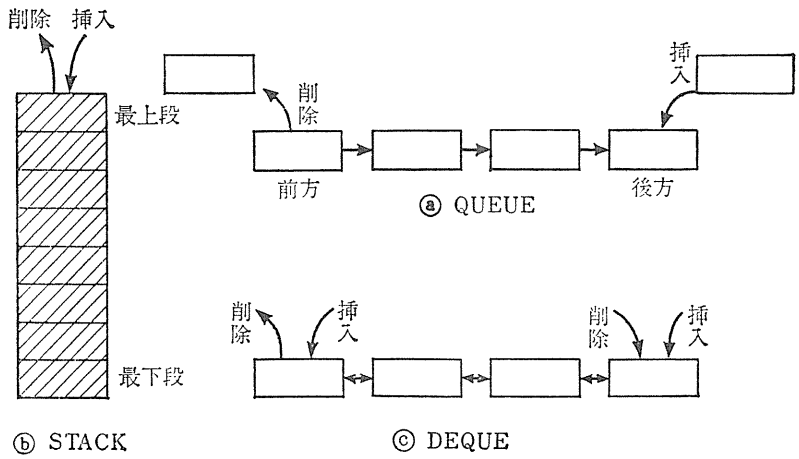
（2）リスト構造の種類

一般にリストを構成する各要素にポインタをどのように連結させるかにより，次のようなリスト構造に分類することが出来る。

〔逐次配置法　Sequential Allocation〕

この配置法は，データ要素間の論理的配列と記憶装置内のワードの物理的配列とが同様な構造をしている配置法である。次の3種類に分けられる。

図2.3　逐次配置によるリスト構造



（1）キュー（queues）

キューとは，一方の端末から入力し，他方の端末から出力する、即ち，リスト要素のそう入，削除の操作がそれぞれ別々の端から行なうリストである(待ち行列ともいう)。一般に削除操作は前方よりのみ行われ，そう入操作は後方から行われる。つまり，そう入，削除を交互に行う場合は右方に移動してゆく。

（2）スタック（Stack）

リスト要素の追加，削除操作がいつも一方向にのみ行われるリストである。すなわち，1個の端末によってアクセスを行い後入先出（last in first out）の意味である。

（3）デキュー（deques）

リスト要素の追加，削除操作がどちらの端末からも行われ，後先入先出(last or first in first out)である。図2.3は逐次配置によるリスト構造を示した。

〔鎖状配置法　Linked Allocation〕

この配置法は，リスト要素間の論理的関係をポインタで連結することである。つまり，どのリストの位置からもそう入，削除を行うことが出来る特徴をもっているので，ポインタの内容によって，単方向リスト（uni-directional list），両方向リスト（bi-directional list）と環状リスト（circular list）に分けることが出来る。

リストは通常その要素間をコンマで区切って，全体をカッコで囲んで表現をするので，その具体的な構造は要素間の論理的関係をポインタで相互連結することによって連結される。カッコが2重，3重になるとそのデータ構造が複雑になる。又その表示方法は1段下げることによって表現される。計算機の記憶構造として考えた場合，アドレス概念を導入して表わすことも可能である。図2.4はその実施例である。この特徴は，隣接同志の駅名が明確に示されていることである。これを図2.4ⓐのように登録することも出来る。下りと上り方面の部分は隣の停車駅の格納番地が入っている。逐次構造では，情報の格納されている番地自身が，隣接の要素の位置を知るための情報になっていたが，ここでは隣接の要素がどの番地にあるかを明確に記憶してある。

図2.4ⓐのようなものは不便なので，ここでは矢印を使用して接続している番地を指すように表わしている（図2.4ⓐ）。このような番地を指す目的で格納されているデータ（番地）をポインタという。その部分は⊠印となっているが，これも一種のポインタであり，リストの終端を示している。

ところで図2.6のように，リスト構造は待ち行列であり，わずかなポインタの書きかえだけですむ。図2.7は文字処理の構造である。この例では“MORIOKA”という文字列をMORIA“とかきかえたときに変化を示している。このリスト構造の1つの箱は，文字の入っている部分とポインタの入っている部分で合成されているが，前者をデータ部，後者をポインタ部ともいう。

図2.7のⓐのリストでは，10000番地を指定すると「MORIOKA」という文字列を表わす。又，

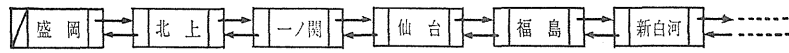
図2.4　計算機による東北新幹線停車駅の表示リスト

	↓下り	↓当駅	↓上り
6	福　島	仙　台	古　川
7	郡　山	福　島	仙　台
8	新白河	郡　山	福　島
9	宇都宮	新白河	郡　山
10	大　宮	宇都宮	新白河

(a) 東北新幹線停車駅の表示板

番地	下り	当駅	上り
500	504	仙　台	496
504	508	福　島	500
508	512	郡　山	504
512	516	新白河	508
516	520	宇都宮	512

(b) 計算機内での格納方式



(c) リストの構造表示

図2.6　待ち行列のリスト

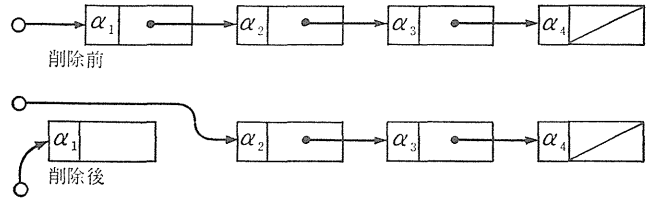
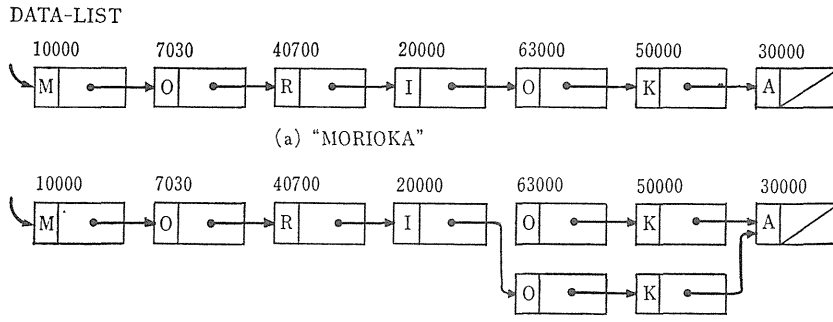


図2.7　ポインタ変化



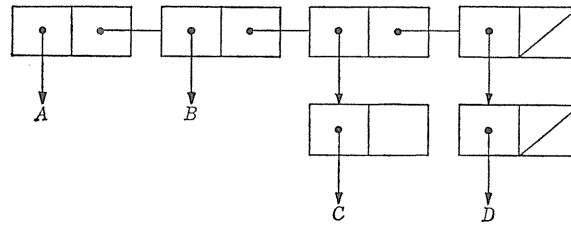
(a) “MORIOKA”

(b) “MORIA”

5000番地を与えると「KA」という文字を表わしている。このように一方向リストの与えられた番地から順々にたどってえられるデータ部の列を引用符（「“」と「”」）で囲ったものをそのリストの表現とよぶ。図2.7ⓑの10000番地の表現は“MORIA”である。図2.8はそのリスト表現である。

最後に，リスト構造の用途は数式処理，人工知能，データベース，文章画像処理，ファイル・システム，オペレーティング・システム，及びシミュレーションなどの非数値計算において広く用いられている。

図2.8　(A,B,(C),(D))のリスト構造



2.3　配列と逐次構造

一般に線形構造や配列などを電算機に記憶させる場合，電算機の記憶場所のそれぞれにつけられた番地をそのまま利用して，その番地の順に記憶していく方法でよく使用されている。図2.9のような成績表の順位では，一番の人は必ずしも一番地に割り当てられるのでなくて，データとして記憶することの出来るような番地から始まるのであり，1人に1番地分確保してあればまに合うという保証はないが，しかし，このテーブルが仮に5000番地分を占めていたとしても，このテーブルを扱うプログラムは面倒にはならない。これは電算機にとって最も特異とする加減乗除の演算だけで番地を計算し，その内容を確認することが出来る。

図2.9　成績表

順　位	名　前	成　績
1	中　村	100
2	久保居	95
3	小　林	87
4	松　木	75
5	大　堀	70

即ち，今N×Mの行列や数列などを記憶する時を考えてみる。例として4×3の行列Aと3×4の行列との積Cを作ることにすると，図2.10のようになる。

Cの要素C_{ij}は，$a_{i1}b_{1j}+a_{i1}b_{2j}$となる。このような場合は，$i$と$j$が与えられた時，行列の要素を記憶してあるところより取り出したり，またはその結果をすみやかに記憶させたりする必要がある。電算機の主記憶装置では，記憶の各要素がとりつけてある。番地が与えることによって，直ちにその内容をとりだし，その内容をかきかえることが出来るので，何らかの方法でiとjを番地に対応させることになるが，次のように各番地を対応させることも出来る（図2.11参照のこと）。

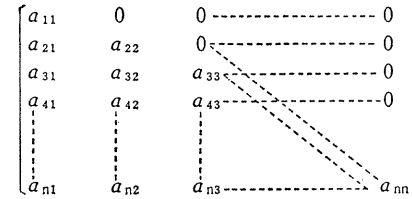


図2.10のAやBを2次元の配列ともいう。1次元の配列とは，a_1，a_2，a_3，……，a_nのような数列である。勿論これらの要素を記憶の各要素に対応させることは容易である。同様に$n_1\times n_2\times$……$\times n_n$の次元の配列を扱うことも出来る。三角行列なども同様な考えで対応させることが出来

図 2.10　配列と番地の対応関係

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \end{pmatrix} \Rightarrow$$

番地		番地	
δ	: a_{11}	β	: b_{11}
$\delta+1$	: a_{21}	$\beta+1$	: b_{21}
$\delta+2$	: a_{31}	$\beta+2$	: b_{31}
$\delta+3$	: a_{41}	$\beta+3$	: b_{12}
$\delta+4$	: a_{12}	$\beta+4$	: b_{22}
$\delta+5$	: a_{22}	$\beta+5$	: b_{32}
$\delta+6$	: a_{32}	$\beta+6$	: b_{13}
$\delta+7$	: a_{42}	$\beta+7$	: b_{23}
$\delta+8$	: a_{13}	$\beta+8$	: b_{33}
$\delta+9$	: a_{23}	$\beta+9$	: b_{14}
$\delta+10$	: a_{33}	$\beta+10$	: b_{24}
$\delta+11$	: a_{43}	$\beta+11$	: b_{34}

図 2.11　三角行列の対応関係

番地			
δ	: a_{11}	δ	: a_{11}
$\delta+1$	: a_{21}	$\delta+1$	: a_{21}
$\delta+2$	: a_{31}	$\delta+2$	: a_{22}
⋮	⋮		
$\delta+n-1$	: a_{n1}	$\delta+\frac{i(i-1)}{2}+j-1$	: a_{ij}
$\delta+n$	: a_{22}		
$\delta+n+1$	: a_{32}		
⋮	⋮		
$\delta+2n-2$	: a_{n2}		
⋮	⋮		
$\delta+\frac{n^2+n-2}{2}$	: a_{nn}	$\delta+\frac{n^2+n-2}{2}$	: a_{nn}
(a)		(b)	

る。

上記のような三角形は図2.11(a)または(b)のように対応させればよい。

2.4　行列線形リスト

一般に k 行 m 列の行列 M_{km} は，

$$M_{km}=\begin{pmatrix} a_{11} & a_{12} & & a_{1m} \\ a_{21} & a_{22} & & a_{2m} \\ \vdots & \vdots & & \\ a_{k1} & a_{k2} & & a_{km} \end{pmatrix}$$

において，これを行順にならべると，

a_{11}，a_{12}，……，a_{1m}，a_{21}，a_{22}，……a_{2m}，a_{k1}，a_{k2}，……，a_{km}

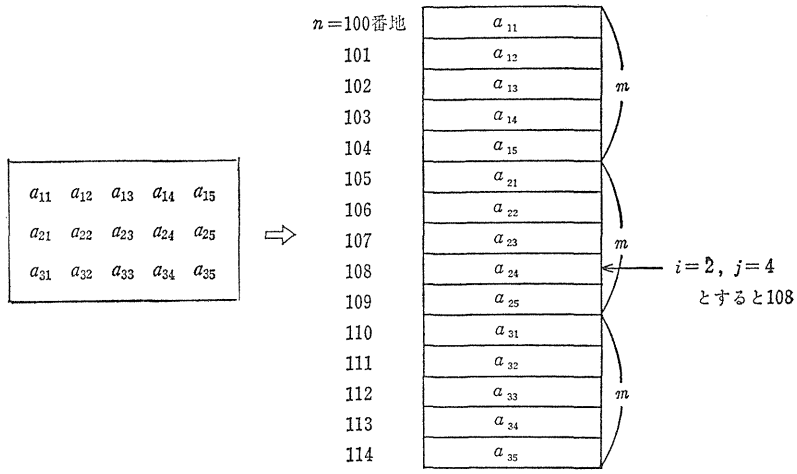
のようになる。A_{ij}が1語データであるとし，a_{11}がn番地であるとすると，データa_{ij}の行順の線形リストにおいて，$n+m(i-1)+j-1$番地になる。$k=3$，$m=5$の場合，図2.12参照。

行順配列 ⟶

a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
a_{31}	a_{32}	a_{33}	a_{34}	a_{35}

$n+5(i-1)+j-1=5i+j+94$

図 2.12　行順線形リスト



この方式では，a_{ij} の番地は k に関係しない。

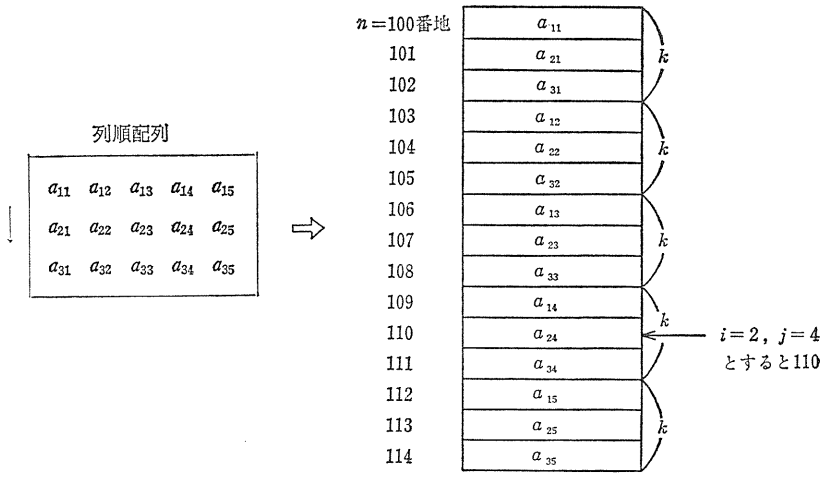
一方，列順線形リストの場合は，前記同様にk行m列の行列を，行順にならべると，a_{11}，a_{21}，a_{31}，……，a_{k1}，a_{12}，a_{22}，……，a_{1m}，a_{2m}，……，a_{2m}，……，a_{km}となる。a_{ij}が1語データであるとし，a_{11}がn番地にあるとき，データa_{ij}は，列順の線形リストにおいて，$n+k(j-1)+i-1$

番地になる。このときはa_{ij}の番地の計算にはmの値は関係しないものである（図2.13参照）。

$k=3$，$m=5$の場合

$$n+3(j-1)+i-1=96+3j+i$$

図2.13　列順線形リスト



尚，固定語長及び可変語長形式の場合のデータ構成は，次のように考えれば良い。即ち，

〔固定語長の場合〕

固定語長形式の場合は，該当データ内の最長のものを基準に考え，例えば全データが10進2桁の数値であるのに，中に5桁の数値データが存在すれば，これをBCDで表現すると，各データは3語を用いて表現しなければならない。従って，データの長さが不均一のときは，固定語長形式では効率が悪くなる（図2.14参照のこと）。

図2.14　固定語長形式

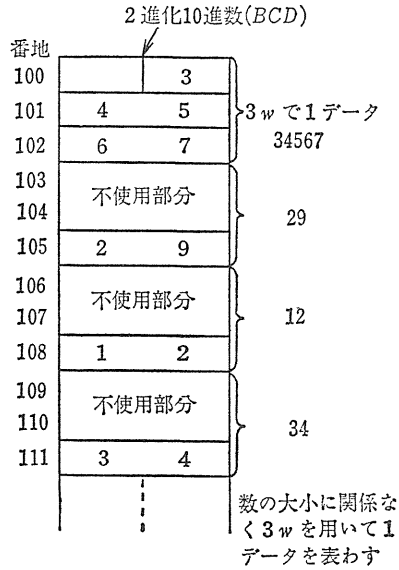
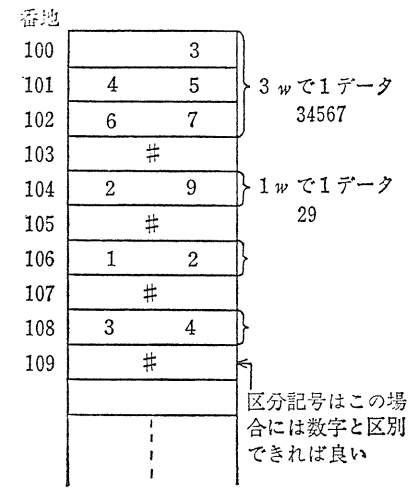


図2.15　可変語長形式



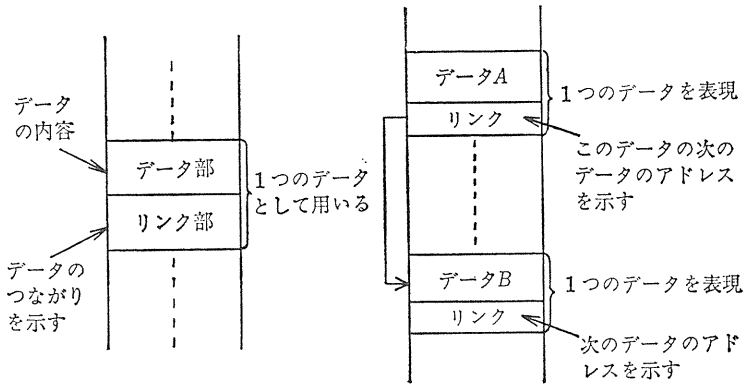
〔可変語長の場合〕

可変語長形式は，不均一な大きさのデータを効率良く記憶させるのに良くもちいられている。

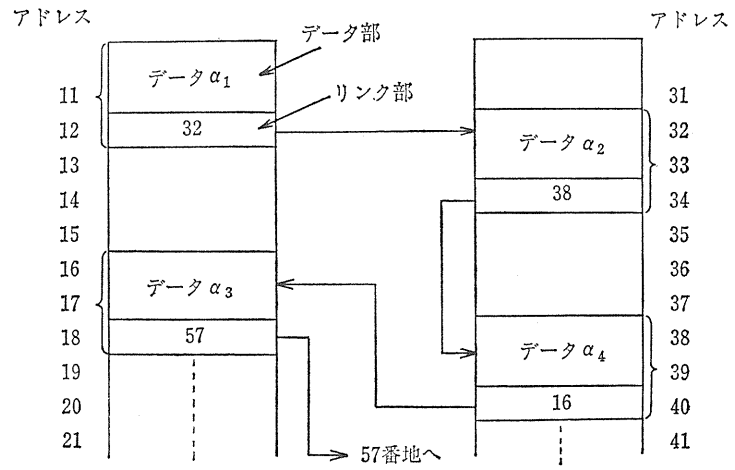
元の配列のi番目のデータが何番地に対応するかは，与えられたデータ集合によってことなるため，データとデータの間には境界となる区分記号を置いて区別する（図2.15を参照）。

2.5　リンクドリスト

線形リストにおいて，もとの配列に対してある規則で決められた順序でデータがならべられている。すなわち，データのアドレス順が配列の順になっている。その方式はデータ自身の記憶を示すデータ部とデータのつながりを示すリンク部を区別して表現する。データ部とリンク部を用いてデータの集合を表現する形式をリンクドリスト（Linked List）という。



データA の次にデータB がある



メモリ内の任意のデータをつなぐことが出来る．

一般に，リンクドリストではつねにリンク部のためのワードエリアが必要である。線形リストと比べた場合は，仕様効率が良くない。しかし，任意の場所のデータを互いに関係づけることが出来るので，線形リストのようなメモリアドレスを連続的に使用しなくてもよいから，リンクドリストを用いてメモリの利用効率を上げることが出来る。

2.6 リスト処理用言語

リスト処理用言語はIPL（Information Processing Language）が最初で，続いてFLPL（Fortran List Processing Language）とLISP（List Processor）がある。リスト処理の基本的な言語はPL/1，ALGOL 68，PASCAL，Cといった汎用プログラミング言語が多い。これらのプログラミング言語についてのリスト操作（又はポインタ操作）は，人工知能や記号処理に使用される。リスト処理用のプログラミング言語の重要性はユーザが，計算機のアドレスのレベルにまで抽象化のレベルを落すことなくアルゴリズムを考えることの出来る点にある。一方，ハードウエアつまり同一多数のプロセッサと大容量メモリなどが，安価で得られるようになると，ハードウエアを高能率に駆動するような言語仕様が再検討されることもありうる。

リスト処理を能率的に処理するには，処理速度の向上と，所要記憶容量の減少などが考えら得る。前者はポインタを高速に実記憶空間のアドレスに変換をし，目的である箱を短いアクセス時間でとり出してくる問題である。後者はポインタをなるべくコード化し，所要記憶容量を減少させる問題である。両者のリスト処理専用アーキテクチャでは，いかに調和をするかなどが一番大切である。

2.7 トリー構造

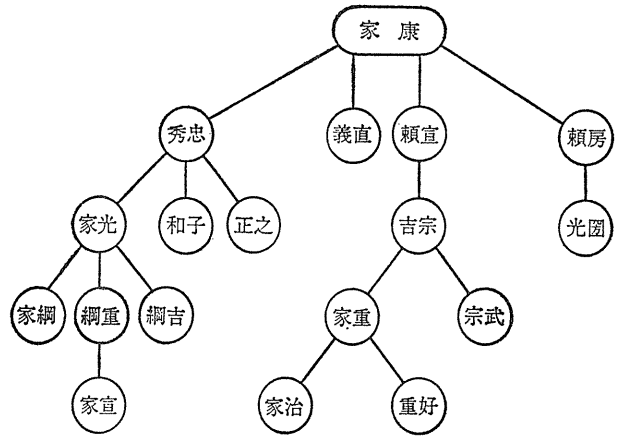
トリー（Tree）とは，一つあるいはそれ以上の枝をもつ節の有限集合である。トリーの出発点となる節をそのルート（根）という。それぞれの枝の末端を葉（Leaf）という（図2.16にトリーの1例を示す）。図中Ⓑ，Ⓒ，Ⓓ，Ⓔ，Ⓕが葉である。

図2.16の(a)の家系図では，家康がこの家系図の根でレベル1である。秀忠以下の家康の4人の子供がレベル2になる。(b)，(c)はそれぞれその例である。(c)のトリーはAからスタートして，次々に葉をたどってゆく。それが右側の葉であればその上の枝に移り，その枝が左側の枝であればそれに対応する右側に移り，又その枝が右側の枝であれば更に上の枝に移る。図2.17はその遷移経過である。これをまとめるとポーランド記法がえられる，即ち，

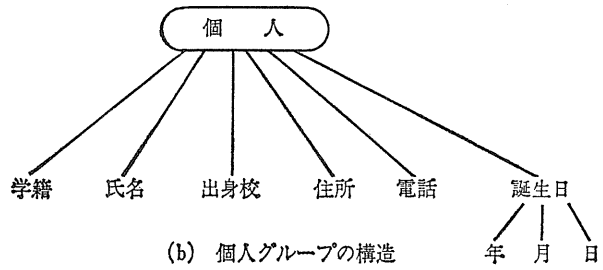
$$ABC * DEF - / +=$$

がえられる。

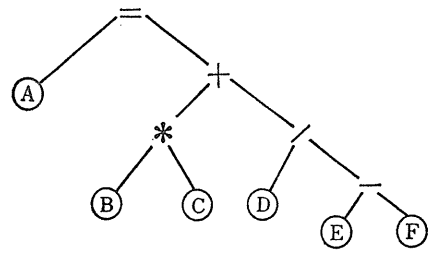
図 2.16　トリー表現



(a) 家 系 図

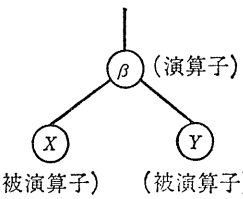


(b) 個人グループの構造



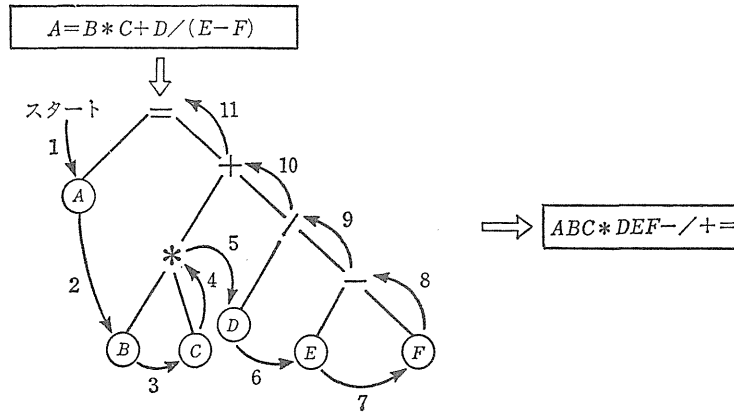
(c) 数式のトリー表現

一般に，トリー構造で表現されている演算方式は，



というパタンのところをさがして X と Y に β の演算をほどこし，その結果を Z で枝に置換し，Z を新たな葉であると考え，このような枝がなくなるまでくりかえせばよい。

図 2.17　トリーの遷移経過



一方，与えられたソース・プログラムからポーランド記法に変換するアルゴリズムとトリー構造に変換するアルゴリズムは次節でのべるが，トリー変換したものから機械語の命令を生成する時は，効率のよいオブジェクト・プログラムを作り出すことが出来る。したがって，タイムシェアリング・システムのような会話型言語では，実行能率よりもコンパイルやデバック能率を重要視する時には，これらをポーランド記法に変換して実行解釈すればよい。又，バッチ・システムのような大きいコンパイラにおいては，オブジェクト・プログラムの効率を重要視するときにも，トリー構造を変換してから機械語のオブジェクト・プログラムを生成すればよい（尚，バッチ・システムの小さいコンパイラで，もとの式から直接機械語のオブジェクト・プログラムを作り出せばよい)。図2.18(a)は文章構造のトリー表，(b)数式構造のトリー表現のその1，(c)はその 2 である。

2.8　トリー構造から機械語への変換

ここでは数式の翻訳について考える。下記の思想はソース・プログラムをトリー構造に変換する時のアルゴリズムである。即ち，

(1)　与えられたソース・プログラム内の全変数は，すべてスタックの中に入れる。

(2)　左カッコの優先順位は，無条件に演算子のスタックに入れる。

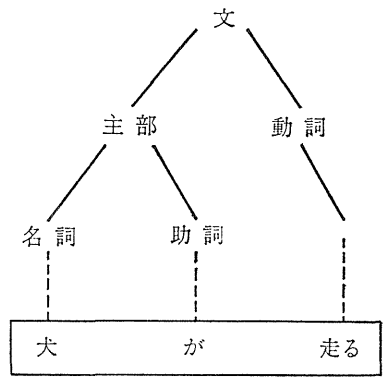
(3)　(2)以外の演算子は，演算子スタックの先頭より優先度が高ければ，そのまま演算子スタックの中に入れる。

(4)　先に入った演算子が数式中の演算子よりも高いか，又は両方の優先度が同一であれば，演算子スタックの先頭の演算子と，変数スタックの上から 2 つの変数をとりだして，トリーの 1 つの節を合成し，その番号を変数スタックの中に入れる。

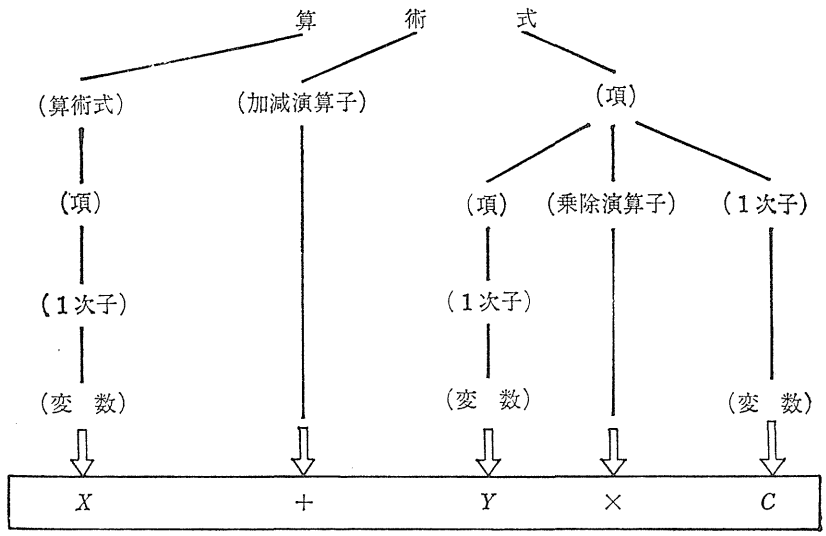
(5)　一方，数式の演算子が右カッコであれば，スタック表面が左カッコの時は両方を抹消して

図 2.18　文章及び数式のトリー表現

(a)　文章構造のトリー表現



(b)　数式構造のトリー表現（その１）



次へ進む。下記はその実施例である（図2.19参照）。

$Z=(A+B)/(C+D)-E*F$

トリーの構成順序は図2.19のようになる。

一方，トリーから機械語への変換は次のように行なわれる。

トリーに変換された数式を機械語命令に変換するには，与えられたトリーをそのルート（Root 根）からスタートして，右まわり評価と左まわり評価について評価をし，両方の枝が葉であるようなノード（節）を変換すればよい。下記はその実施例をシステムモデルを用いて評価をする。

(c) 数式構造のトリー表現（その2）

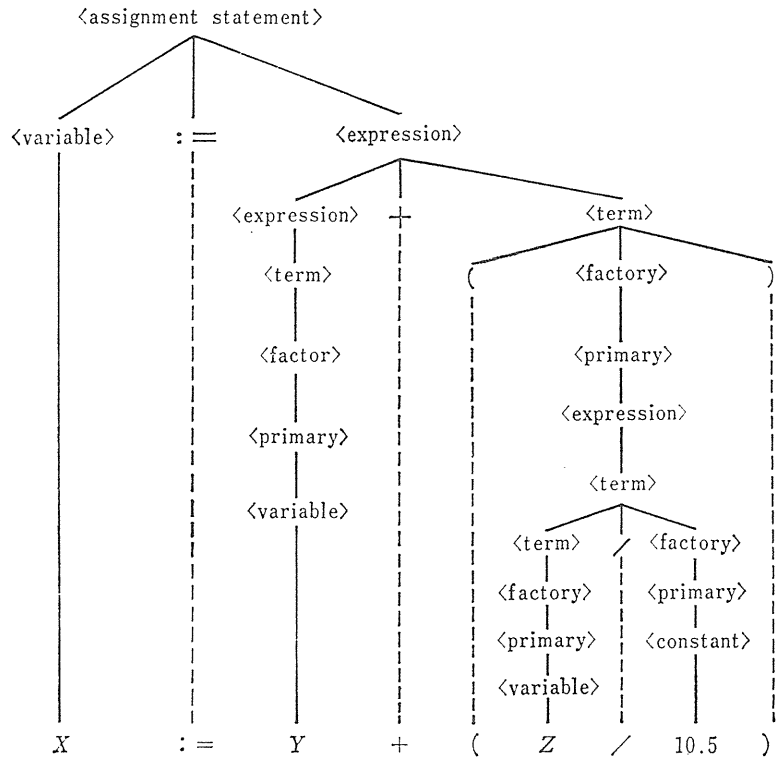
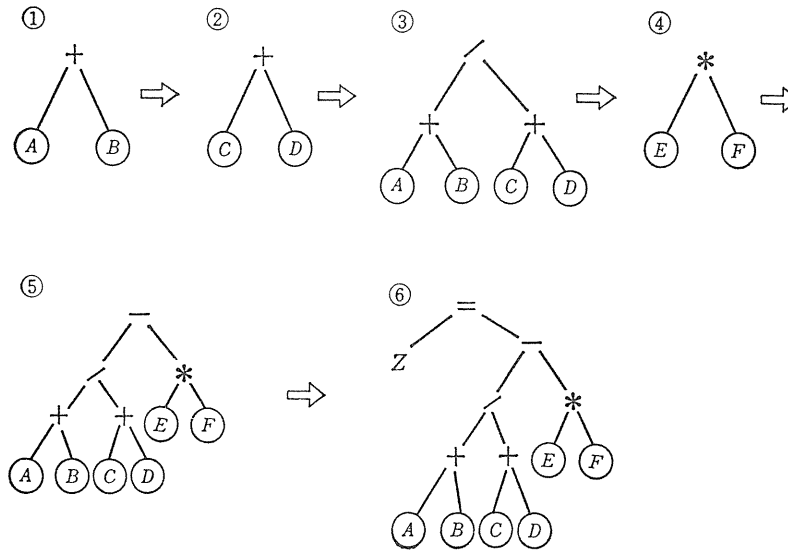
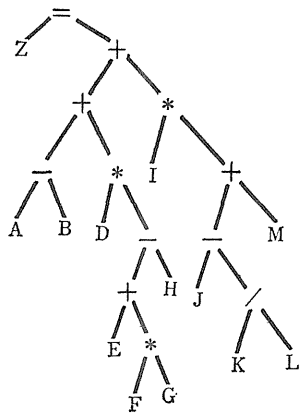


図 2.19　トリーの構成順序



〔例１〕 $Z=A-D+D*(E+F*G-H)+I:(J-K/L+M)$



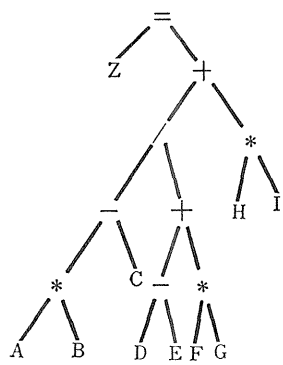
〔右まわり評価〕

MV_i	$=0$,	R_4
MV	K,	R_5
D	L,	R_4
MV	J,	R_5
S	R_4,	R_5
A	M,	R_5
MV	R_5,	R_4
M	I,	R_4
MV	R_5,	Z
MV	F,	R_4
M	G,	R_4
A	E,	R_5
S	H,	R_5
MV	R_5,	R_4
M	D,	R_4
A	A,	R_5
S	B,	R_5
A	R_5,	Z

〔左まわり評価〕

MV	A,	R_4
S	B,	R_4
MV	R_4,	R_5
MV	F,	R_4
M	G,	R_4
A	E,	R_5
S	H,	R_5
MV	R_5,	R_4
M	D,	R_4
A	R_5,	T_s
MV_i	$\doteqdot 0$,	R_4
MV	K,	R_5
D	L,	R_4
MV	J,	R_5
A	M,	R_5
S	R_4,	R_5
MV	R_5,	R_4
M	I,	R_4
A	R_5,	Z

〔例２〕 $Z=((A*B-C)/(D-E+F*G)+H*I)$



〔右まわり評価〕

MV	H,	R_4
M	I,	R_4
MV	R_5,	TS
MV	F,	R_4
M	G,	R_4
MV	R_5,	TS_1
MV	D,	R_4
S	E,	R_4
A	TS_1,	R_4
MV	R_4,	TS_2
MV	A,	R_4
M	B,	R_4
S	C,	R_5
D	TS_2,	R_4
A	TS,	R_4
MV	R_4,	Z

〔左まわり評価〕

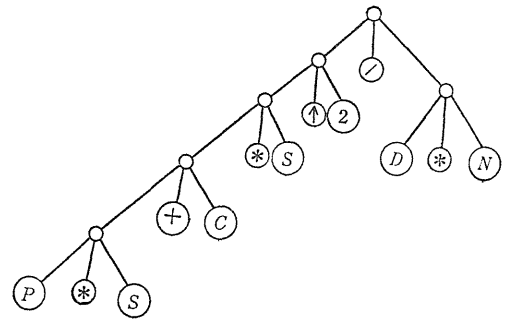
各自検討してみて下さい

以上の様に，右まわりの評価方法のほうがオブジェクト・プログラムの能率が良いことがわかる。左まわりの評価は評価途中の結果の格納命令を省略出来るのは，トリーの根のところと，右側の枝が葉である時の枝の評価の途中結果だけであるが，右まわりの時は，左の枝の途中結果が，そのまま次の節に利用することが出来るので，作業番地に格納する必要がなくなるからである。このような効果は数式が複雑になればなるほど大きい。

2.9 変数値をもった算術式のトリー表現

ここでは，N個のあいことなるリスト（又は数列）について考える。

$$((P*S+C)*S)\uparrow 2/(D*N)$$



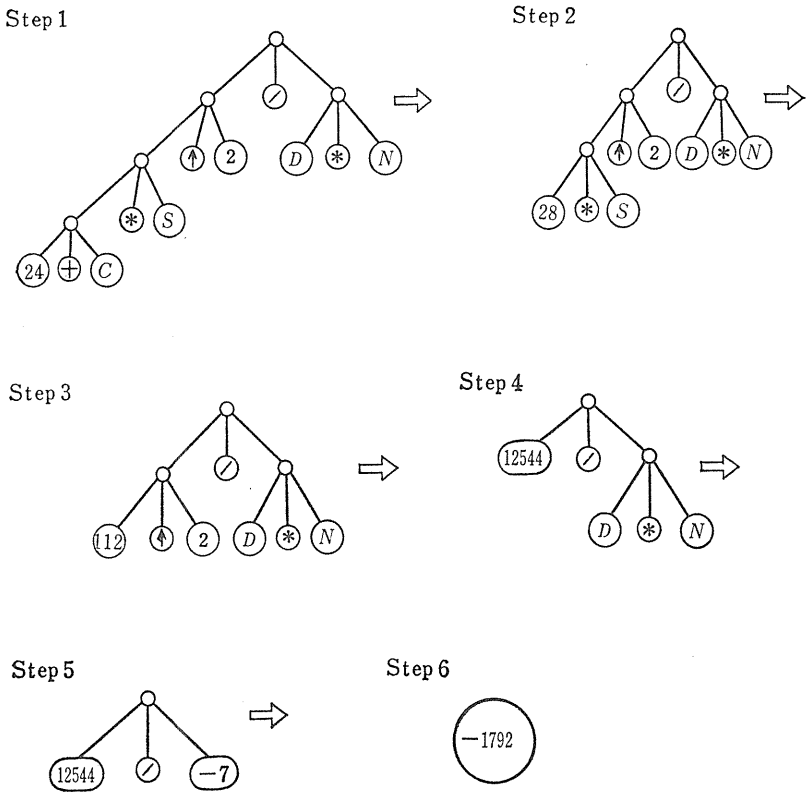
各変数値を次のように仮定しますと，上記のトリーは下記のようになる。

$P=6$，$S=4$，$C=4$，$D=-1$，$N=7$

前述のようにトリーにはルートがある。そこから2つ以上の枝が出てほかの節に伸びていく。そうして，次々と枝わかれして最終的には端末の節または葉に到達して終る。枝につながる節には端末であるかまたは根にする。節にはレベルによって位置づけすることが出来る。レベルは根からその節までの路にそった節の数である。

次に，トリーの各変数にそれぞれ値を与えれば，トリー全体が単一の葉で置きかえることが出来る。すなわち，1つの部分トリーが1つの演算子と2つの被演算子で構成されているとき，その部分トリーを1つの葉で置きかえることが出来る。

次に，記憶装置でのトリーの表現についてのべる。下記に挙げた実例は，それぞれの節に番号をつけ，節jに入っている情報は次のようなトリー表現で容易にみつけられる。即ち，MORIiには節iの内容がストアされている。TOKOYの行iには節iの子供の節の番号が入っている。S_iにはTOKOYの行iのブロック，すなわち節iの子供の数が入っている（図2.21）。図2.21の中，7番目の節は文字“G”をもち，又3人の子供を，すなわち節13，14，15をもつことがわかる。この実施例では，節には番号がついていて，レベルごとに左から右へ順に表に入っているけれども，



必ずしもこのようにしなくてもよい。

一方，電算機において，算術式を機械語命令に変換するのは大変面倒である。とくに次のような場合である。

$A*X^2+B*X+C=0$

の２次方程式について考えると，

$$\pm\frac{B+\sqrt{B^2-4*A*C}}{2*A}\longrightarrow\frac{B-\sqrt{B^2-4*A*C}}{2*A}$$

ここでは，単項の負符号があると複雑になるので，負符号をとりのぞいた。

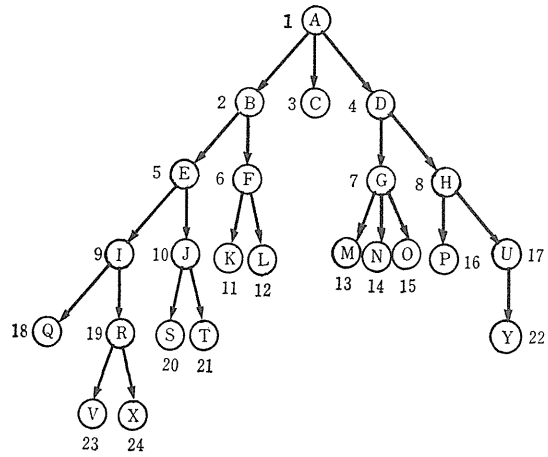
$(B+(B\uparrow 2-4*A*C)\uparrow 5)/(2*A)$

にして，これをポーランド記法に変換する。

2.10　リスト処理とアーキテクチャ

計算機のアーキテクチャの決めかたは，計算機の設計にたずさわる者にとって最も大切な問題である。計算機の発展初期には，設計者は大からにアーキテクチャを決定することが出来た。し

図 2.21　トリーの記憶表現



	MORI		S		TOKOY		
1	A	1	3	1	2	3	4
2	B	2	2	2	5	6	
3	C	3	0	3			
4	D	4	2	4	7	8	
5	E	5	2	5	9	10	
6	F	6	2	6	11	12	
7	G	7	3	7	13	14	15
8	H	8	2	8	16	17	
9	I	9	2	9	18	19	
10	J	10	2	10	20	21	
11	K	11	0	11			
12	L	12	0	12			
13	M	13	0	13			
14	N	14	0	14			
15	O	15	0	15			
16	P	16	0	16			
17	U	17	1	17	22		
18	Q	18	0	18			
19	R	19	2	19	23	24	
20	S	20	0	20			
21	T	21	0	21			
22	Y	22	0	22			
23	V	23	0	23			
24	X	24	0	24			

かし，ソフトウエアの開発に膨大な人手と経費が必要なことが明らかであり，ソフトウエアの開発が増すにつれて，アーキテクチャが固定化する傾向がでてきている。特に，従来のアーキテクチャを包含するようなアーキテクチャであれば，ソフトウエアの蓄積を生かすことが出来る。また新たに開発されている応用分野には，新しいアーキテクチャの計算機が要求される。

一方，アーキテクチャの変更を要求するインパクトはハードウエアの技術の進歩，ソフトウエアからの要求，応用面の拡大にともなう要求といったいろいろな分野から発生する。

リスト処理の効率を高めるには，処理速度のレベルアップと，所要記憶容量の減少がある。前者は，ポインタを高速に実記憶空間のアドレスに変換し，かつ，目的とする箱を短いアクセス時間でとり出してくる問題である。後者は，ポインタを出来るだけコード化して，所要記憶容量を減少させる問題である。両者は，トレードオフの関係にあるため，リスト処理専用アーキテクチャでは，両者をいかに調和させて，実現するかが，一つの問題である。

注：本稿は富山鐵朗，渋井二三男著“電子計算機のシステムプログラム”を加筆修正したものであり，本稿執筆にあたり富山鐵朗氏のご指導をいただいた。ここに深謝する。

参考文献

1. 電子計算機の構造とアセンブラに関するもの

（1）石上・成毛共著『計算機の基本動作』富士学院，pp.17～22，pp.39～53
（2）マニュアル『FACOM 230-25 FASP 文法書』富士通（株）
（3）マニュアル『FACOM OS Ⅳ』アセンブラ文法書，富士通（株）
（4）石井康雄著『コンピュータ入門』オーム社
（5）『System/360 入門－1』日本 IBM（株）教育部
（6）IBM System/360 Principles of Operation File No. 360-01，Form A22-6821- 7
（7）ドノバン著『システム・プログラムⅠ・Ⅱ』コンピュータ・サイエンス，pp.24～63
（8）岸田孝一著『システム・プログラム入門』日本経営出版会，pp111～146
（9）Fisher, F. and Swindle, G., Computer Programming Systems, Holt, Rinenhert and Winter, 1964（浦　昭二訳）
（10）マニュアル『PANAFACOM U シリーズ』UMOS アセンブラ文法書，富士通（株），pp.3～28
（11）マニュアル『PANAFACOM U-1200/1400/1500』ハードウエア解説書，富士通（株），pp.11～12
（12）小出寿夫著『アセンブラ　Ⅰ，Ⅱ，Ⅲ』PANAFACOM U-1400
（13）渡部弘之著『コンピュータ設計技術（Ⅱ）』CQ 出版社

2. リスト処理とアーキテクチャ

（1）A. Newell and F. M. Tonge. “An introduction in information Processing, Language V.”, Communication of the ACM, Vol.3, pp. 205～211, 1960
（2）小岩　明著『リスト処理』（情報処理技法講座）富士学院，pp.9～50
（3）I. フローリズ著・久保寛彦訳「データの構造」竹内書房，pp.128～164
（4）Victor H. Yngue, “COMIT as an IR”, ACM, Vol. 5, pp. 19～28
（5）J. Weigenhaum, “Symmetric List Processor”, Communication of the ACM, Vol. 6，1963, pp.524～536
（6）情報処理特集『記号処理と計算機アーキテクチャ』Vol. 23, 1982, No. 8, pp. pp.712～718，社団法人情報処理学会