47—S005

汎用グラフィック言語の開発

—ディスプレイ・システムの研究開発—

昭和48年3月

JIPDEC

財団法人 日本情報処理開発センター

この報告書は日本自動車振興会から競輪収益の一部である機械工業振興資金の補助を受けて昭和47年度に実施した「汎用グラフィック言語の開発」の成果をとりまとめたものであります。

序

当財団は，情報処理に関する研究開発の一環として情報処理システムにおけるるグラフィック・ディスプレイの利用に関する各種ソフトウエアの研究開発を進めております。

最近各方面でグラフィック．ディスプレイ装置の利用が普及しつつありますが，この装置を利用した各種アプリケーションをささえるソフトウエアの開発がたちおくれ，ゆき悩んでいる面が多々うかがわれます。これを解決する手段の一つとして汎用グラフィック・ソフトウエアの整備，特に平易な表現でグラフィック処理過程を記述し得る，ハイレベルの汎用グラフィック言語の開発が期待されております。

本報告書は，当財団で開発した「汎用グラフィック言語」の開発成果を述べたものであります。

本書がこの方面に関心のある方々に広く利用され，わが国情報処理技術発展の一助として寄与できるよう念願いたす次第であります。

昭和４８年３月

財団法人　日本情報処理開発センター

会長　難　波　捷　吾

ま え が き

グラフィック・ディスプレイはマン・マシン・コミュニケーションの接点として注目されているが，ハードウエアのコスト高，汎用グラフィック・ソフトウエアの不足などにより一般的に広く利用される段階にはまだ到っていない。

グラフィック・ディスプレイの機能を効果的に利用するための汎用グラフィック・ソフトウエアは次の3つのレベルに分類される。

(1) Graphic Operating System

(2) Graphic System Program

(3) Graphic Application Program

このうちGraphic System ProgramはGraphic Operating Systemの制御のもとで図形処理あるいはインタラクティブ処理などのグラフィック・システムにおける基本的な機能の一切を処理するシステムである。Graphic Application Programは問題向きのプログラムであるが，通常このGraphic System Programのサポートを受ける。Graphic System Programの最も汎用的なアプローチの一つとしてまずグラフィック言語が考えられる。グラフィック・ディスプレイの存在を前提としたインタラクティブ・システムにおけるプログラミング言語としてはすでに各種のものが開発されているが，コンパイラ言語のレベルに属し，しかもグラフィック・データの扱い，データストラクチャ操作，割り込み処理機能の全てを備えている言語は少ない。

当財団では，このような背景の元でグラフィック・ソフトウエアの体系化を指向し，汎用グラフィック言語の開発およびグラフィック・システムのさらに効果的な利用をめざしている。４５年度はマルチ・プロセッサの下でのグラフィック・オペレーティング・システム（CGOS）を開発し，４６年度，４７年度の２カ年計画によりCGOSの下で嫁動する汎用グラフィック言語 UNGL（UNiversal Graphic Language）を開発した。４６年度の報告書に於ては既存のグラフィック言語の概観と，UNGLの言語仕様の大要を述べたが，こ

の報告書ではUNGLのインプリメンテーションを中心に報告する。

本書は5章に分かれており，第1章ではUNGLの概要及びシステム構成，プログラム構成について述べ，第2章ではUNGLプリ・コンパイラの機能及び処理について述べる。第3章以下でグラフィカルな機能すなわち図形データ処理機能，割り込み処理機能，汎用データ・ストラクチャ操作機能等につきその処理アルゴリズムおよびインプリメンテーションの立場から見た機能のやや詳細な説明等を行っている。

目　　次

1. UNGLの開発

1. UNGLの開発

1．1　グラフィック言語

UNGL（UNiversal Graphic Language）はグラフィック・ディスプレイ装置を利用する多種のアプリケーション・プログラムの作成を容易にする事を目的として作成された汎用グラフィック言語である。以下にUNGL作成の前提となるグラフィックスの現状すなわちグラフィック・ソフトウエアの現状とグラフィック言語のニーズを述べ，さらに高度なグラフィック・プログラムの概念を明確化し，UNGLの全体像について述べる。

1．1．1　グラフィックスの現状

コンピュータ・グラフィックスの実用目的は，グラフィック・ディスプレイ装置をマン・マシン・コミュニケーション（Man Machine Comunication）の道具として利用する事にあると言われ，CAD（Computer Aided Design），構造解析，グラフィックARTをはじめ，コマンド・コントロール，CAI（Computer Assisted Instruction），IRなどの広範な分野でインタラクティブ・システムとして注目されている。しかし，この様なシステムをささえるグラフィック・ソフトウエアの体系化が完全にはなされておらず，しかもハードウエアのコスト高という事も相まって，一般に広く利用される段階にはまだ到達していない。

ここでグラフィック・ソフトウエアの持つべき機能について整理すると次の通りである。

(1)　割込み処理とプログラム制御機能

(2)　図形表示及び入出力機能

(3)　図形データ，問題データの処理機能

(4)　マン・マシン・コミュニケーション機能

(5)　ハイレベルな図形処理用プログラム

このうち(1)，(2)は対象とするハードウエアにある程度依存する機能であり，通常グラフィック・オペレーティング・システム中に含まれる機能である。

(3), (4), (5)のソフトウエア機能は完全には体系化されていないのが現状であり，それらの汎用化は困難であるとされている。

この様なグラフィック・ソフトウエアの環境化でグラフィック言語は次の様な状況にある。

a．グラフィック言語のニーズ

グラフィック・ディスプレイは，マン・マシン・インタラクティブ・システム（man-machine interactive system）として多大の効果を発揮するが，そのためのアプリケーション・プログラムの作成には種々の問題点がある。例えば，グラフィック・アプリケーションに於ては一般のコンパイラ言語では記述しにくい図形データ処理，インタラクティブ処理，汎用データ・ストラクチャ操作等が重要な役割を果たすが，これらの扱いにはハードウエアおよびソフトウエアの専門的な知識を必要とする。一般にグラフィック．アプリケーション・システムの使用者はコンピュータの専門家であるよりはむしろ航空機や自動車の設計技術者，あるいは回路解析の専門家などであり，彼らがそのシステムの作成自体に関係する機会も多い。従って上記のような複雑な処理を簡単に記述し得るハイレベルのグラフィック言語の出現が大いに期待されている。

b．グラフィック言語の性格

UNGLはプロセデュラル型記述式言語を指向しているが，その理由及び言語の分類について以下に示す。

グラフィック・システムにおけるハイレベル言語には種々のタイプのものがある。それらを分類すると先ずプロブレム・オリエンティドなパラメトリックなものとプロセデュラルの2種類があり，さらにその表現の形式としてはステートメントの記述による記述式と，ファンクション・キーやライトペン操作でステートメントを指定する指示式とがある。グラフィック・システムの理想的な形態としては，一見パラメトリックな指示式の言語が想定されるが，実用上からは，たとえプロブレム・オリエンティドな言語であっても指

示式のみに限定されるのは不便な事もあり，記述式，指示式の両方の形式が適材適所にうまく利用できる事が望ましいとされている。指示式と記述式とは利用上は性格が異なるが，実際には同一言語の二つの形式であり，この間に１対１の対応をつけることができる。従って言語自体としては，記述式を基本に考えておけば十分であろう。またプロブレム・オリエンティドな言語の作成には，プロセデュラル言語を利用する事もできる。従って上記の4種のタイプのうちプロセデュラル型記述式言語が最も基本的なものであり，他の言語はそのアレンジ内至はややマクロ化した機能として比較的容易に拡張可能である。

１．１．２　グラフィック・プログラムの概念

グラフイック・プログラムは，与えられた問題処理をマン・マシン・インタラクティブに行う必要がある。この事はＣＲＴに表示された図形が問題処理を行なう上で重要な役割を持つ事を示す。

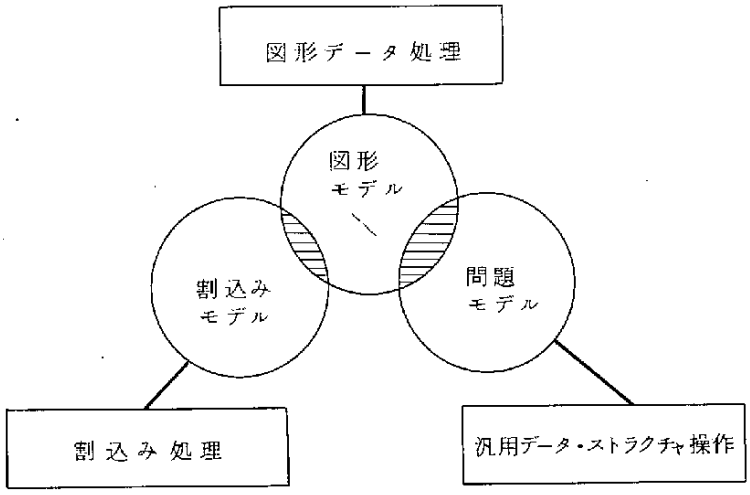
Ｃ．Ｉ．Johnsonは図形のＣＲＴ表示には，２つのモードの実現が考慮されるべきであるとしている。その１つは，プログラムによる構造化されたデータの解析及び操作の後にその一部が明示的な指定によってＣＲＴに表示されるモードである。もう１つのモードは，プログラムによるデータ構造及びグラフィック変数の操作の側面効果として，その図形的変化が刻々とＣＲＴ上に表示されるモードである。すなわち，ＣＲＴを１つの媒体として，インタラクティブな処理が行なえる事と，与えられた問題の解析が充分に行なわれ，その効果としてＣＲＴ上の図形変化が可能な事が，グラフィック・プログラムに課せられた命題である。

この様なインタラクティブ・システムを指向する高度なグラフィック・アプリケーション・プログラムにおいては，図形データ処理の対象となる図形モデル，割込み処理の対象となる割込みモデル，汎用データ・ストラクチャ操作の対象となる問題モデルの概念を導入する事によって，プログラムの論理体系を明らかにする事ができる。すなわち各処理内容をモデルによって表

現する事により，高度なグラフイック・アプリケーションを論理付けてとらえる事ができる。

以下に各モデルについて述べるが，３つのモデル間の関係は〔図１－１〕の通りである。

〔図１－１〕　モデルの関係

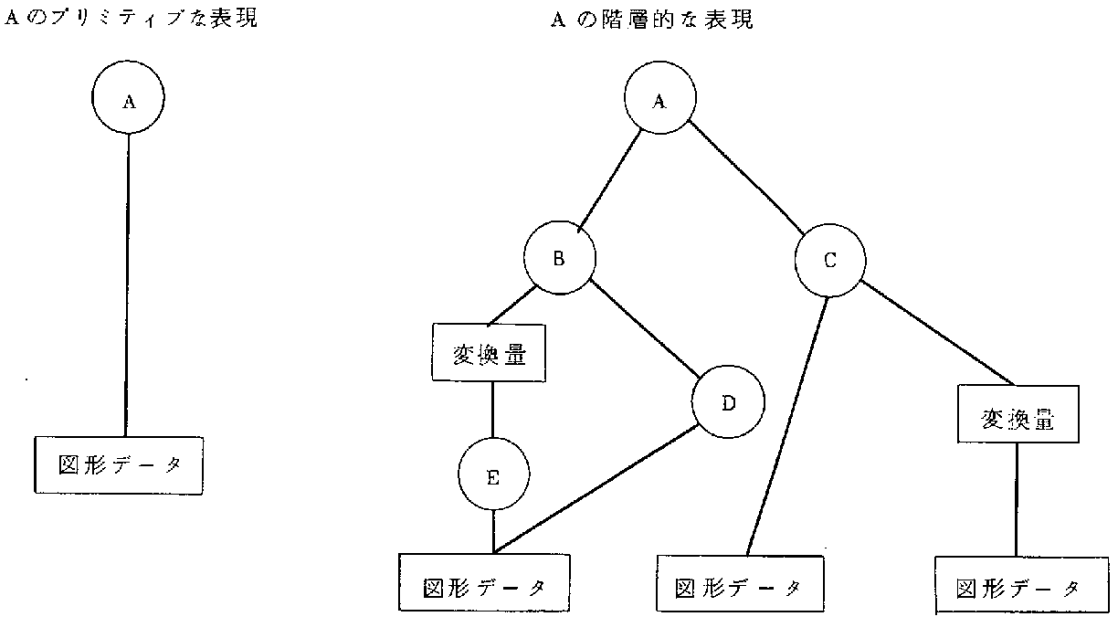


a．図形モデル

図形モデルは基本的にはＣＲＴ上に表示される図形そのものであり，幾何学的情報を内容として持つ。すなわち，図形要素とそれに関する変換量，例えば，回転，移動，拡大，縮少，透視，投影等の情報量を含んだものであり，図形の実態と，図形を識別するデータとからなる。

一般に図形モデル内のデータ構造は，クロス・コネクションを扱ったトリー構造となっている。〔図１－２〕に図形モデルの表現を示す。

〔 図 1 － 2 〕　図形モデルの表現

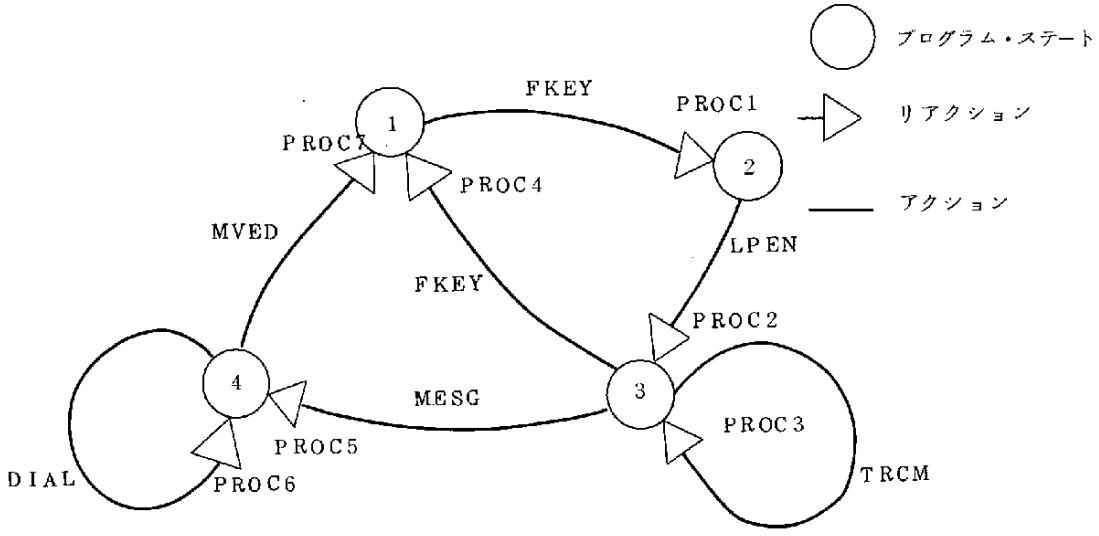


b．割込みモデル

グラフィック・アプリケーション・プログラムにおいては，割込み処理過程が非常に複雑である。すなわち，プログラムの状態遷移がダイナミックに行なわれるので，その遷移過程を知る事が必要となる。この様な割込み過程の把握は，ユーザからのアクションによるリアクションと，その後のプログラム状態という概念すなわち，有限オートマトンとしてとらえるのが適切である。

割込みモデルでは，プログラム遷移をプログラム・ステートとしてとらえる。〔図１－３〕に割込みモデルを示す。

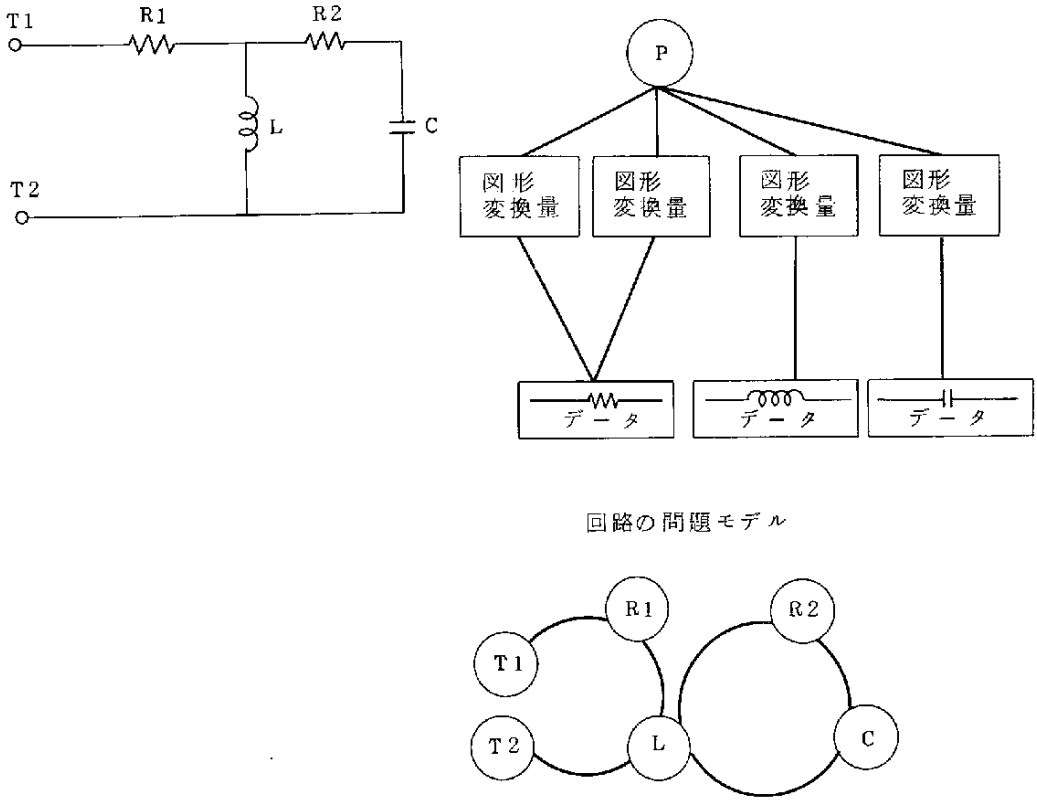
〔 図 1 － 3 〕　割り込みモデル



c．問題モデル

グラフィック・アプリケーション・プログラムで，与えられた問題を解決するために，問題をモデルとして扱う事が考えられる。例えば〔図１－４〕の様な電気回路を考える時，図形表現は，単なる幾何学的表現のみで可能であるが，ユーザが欲するのは，図形そのものよりも，その中に含まれる物理的な性格と考えられる。すなわち与えられた問題解決のためのモデルは，図形とは直接関係のないインピーダンス（レジスタンス，インダクタンス，キャパシタンスの関係）といったもので表現されるべきである。このように問題の本質を考える時，問題モデル表現の必要性が生じる。

〔図１－４〕　図形モデルと問題モデル



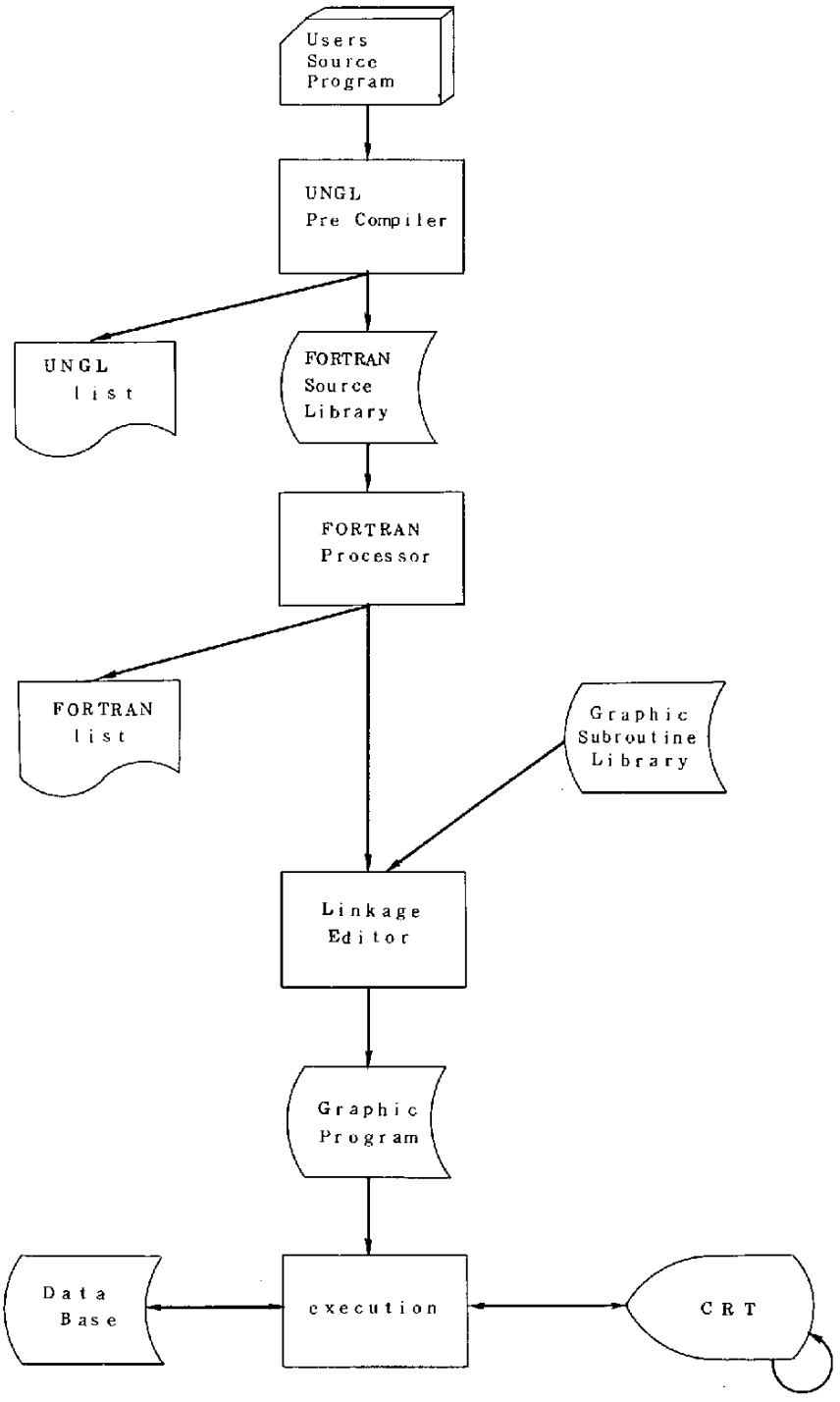
１．２　ＵＮＧＬの概要

UNGLは，図形モデル，割込みモデル，問題モデルの概念を明確化し，グラフィック・ディスプレイの高度な機能を簡単な記述で利用し得る事を目的とした言語で，FORTRANをベース言語とし，それにグラフィックス特有の機能拡張を加えたものである。

UNGLは①プリ・コンパイラと②グラフィカル・モジュールの２つに大別される。前者はグラフィック・アプリケーション・プログラムの評価とFORTRANソース・プログラムの作成を主な目的とし，後者は，ユーザのグラフィック・アプリケーション・プログラムの実行にともなって，グラフィカルな機能を発揮する事を目的としたランタイム・モジュールである。

〔図１－５〕にUNGLの全体像及び処理手順を示す。

〔図１－５〕　ＵＮＧＬの処理手順



UNGLの特徴を最も象徴するのは，グラフィック・サブルーチン・ライブラリである。グラフィックス特有の問題すなわち図形モデルの扱い，問題モデルの扱い，割込みモデルの扱い等の複雑な処理は，グラフィック・ランゲージ・ファシリティに付帯して提供されるサブルーチン・ライブラリを呼び出す事によって果される。

すなわち，ユーザがグラフィックス特有な機能を簡単なステートメントによって要求する事によって，必要に応じて機能を果すべきモジュールが，ユーザのプログラムにつけ加えられる。

次にプリ・コンパイラの特徴及び，グラフィカル．モジュールの特徴について述べる。

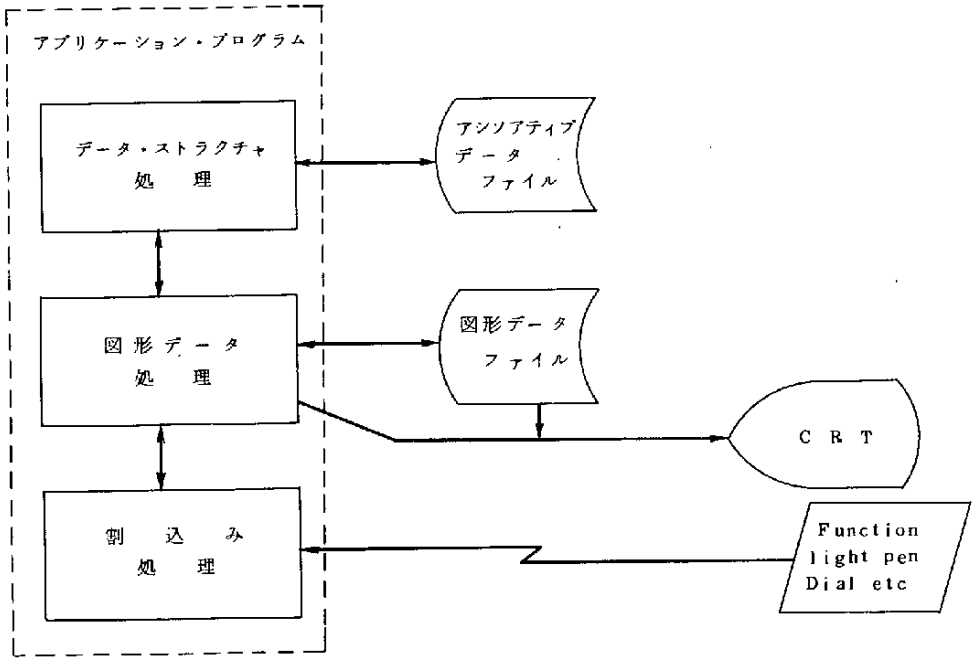
1．2．1　UNGLプリ・コンパイラの特徴

UNGLプリ・コンパイラは1パス（Pass）のコンパイラで，UNGLのソース・ステートメントの解読及びFORTRANソース・ステートメントの作成を行ない，同時にFORTRANコンパイラの制御，リンケージ・エディタの制御に関する情報についても処理する。

1．2．2　グラフィカル・モジュールの特徴

UNGLのグラフィカル・モジュールは(1)図形データ処理，(2)割込み処理，(3)データ・ストラクチャ処理の3つに大別され，それらのモジュールの機能を以下に示す。なお，各処理モジュールの関連を〔図1－6〕に示す。

〔図１－６〕　グラフィカル機能関連図



A．図形データ処理機能の特徴

①　図形データの概念が明確

図形に関係する量を４つのデータ・タイプとして表現し，その概念上の区別を明確化している。４つのデータ，タイプとは，イメージ，イメージ・データ，図形変換マトリックス，論理化因子である。

②　３次元図形の扱いが可能

２次元イメージによる２次元図形の扱いと同様に，３次元イメージによる３次元図形の扱いも可能である。

③　図形操作が容易

図形の併合，代入，回転，移動，スケーリング，透視，投影，グルーピングなどの図形操作を容易に実現できる。

④ 表現形式が優れている

複雑な図形を，イメージ・エクスプレッション中に端的に表現する事ができる。

⑤ 図形表示の管理機能を持つ

表示される画面はフレームという単位で管理され，保存しておくことができる。

⑥ 画面操作が可能

フレームに出力された図形に対して，消去，変更，などの各種の操作を加える事ができる。

B．割込み処理機能の特徴

① プログラム・ステートの概念で取り扱うことが出来る。

アクション，リアクションによって定義される割込み処理過程をプログラム・ステートの遷移という形式で表現することができる。すなわち複雑な割込み処理過程を表現する場合に，その流れをマクロに把握出来る。

② プログラム・ステートの動的変更が可能である。

従来のステート・ダイヤグラム・アプローチでは，プログラム・ステートがプログラム作成時にスタティックに決定されてしまうのに対し，UNGLでは，これを動的に再定義する事が可能である。

③ 通常の割込み処理方式が使用可能

通常の割込み処理の概念をそのまゝ使って，プログラム・ステートを意識しないでも割込み処理を行なう事が可能である。

④ 非同期処理が可能

リエントラント構造を備えたユーザの割込み処理ルーチンに対しては，非同期処理が認められる。

C．データ・ストラクチャ処理の特徴

① LEAPタイプの連想記憶機構を採用している

関連データの扱いに関して，特に一般性を考慮してLEAPタイプの連想

記憶機構を用い，問題モデルの表現を単に本質的な部分だけの記述で可能としている。

② 図形モデルと問題モデルの関連が簡単につく

図形データに表現される論理化因子によって，図形構造と，問題用データ構造の関連がつけられる様になっている。

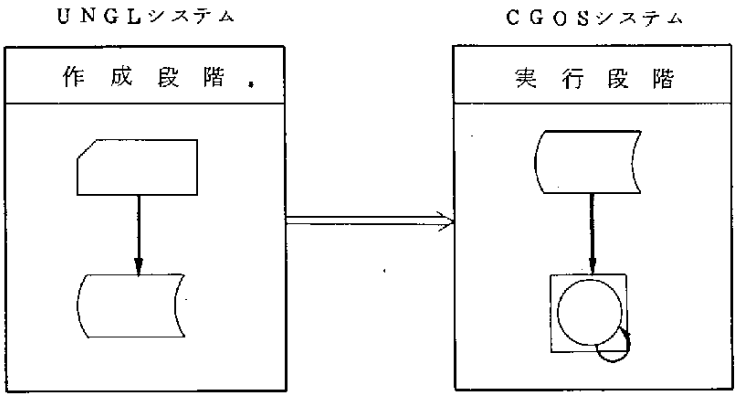
③ 二次記憶装置へのアクセスが可能

汎用のデータ・ストラクチャに於ては，ハッシュ技法に基づいた，連想記憶方式が使われ，効率，二次記憶への拡張性という点で，最もすぐれている。

１．３　システム構成

グラフィック・アプリケーション・プログラムの作成，実行に際して，グラフィック・システムは〔図１－７〕の様なシステム構成を持つ。

〔図１－７〕　グラフィック・システム



UNGLシステムは，UNGLソース・プログラムで記述されたグラフィック・アプリケーション・プログラムをUNGLプリ・コンパイラによって実行可能なプログラムへの変換を行なう。CGOSシステムでは，実行可能なアプリケーション・プログラムのCRTからの起動，実行の管理を行なう。すなわち，利用上では，UNGLのシステム構成とCGOSのシステム構成は切り離されたものと見なすことが出来るが，グラフィック・システムという観点からみると，どちらも，グラフィックスの高度な機能に関してのサポートシステムである。

以下に，CGOS，UNGLのハードウェア構成，ソフトウェア構成について述べる。

１．３．１　ハードウェア構成

A　CGOSのハードウェア構成

機器構成を〔図１－８〕に示す。

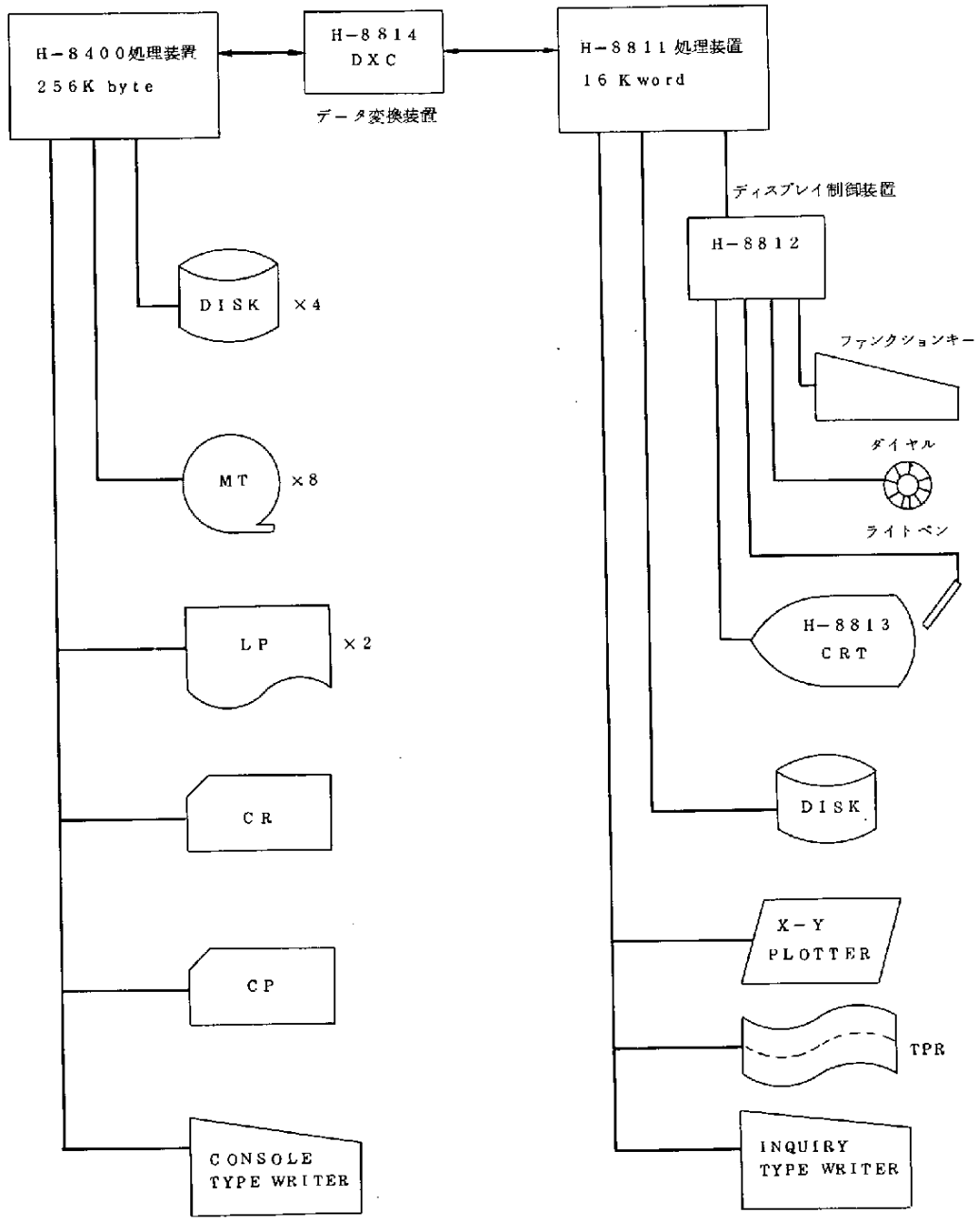
HITAC−8400処理装置をホスト・コンピュータとし，HITAC−8811処理装置をサテライト・コンピュータとする2プロセッサで構成され，2台の処理装置の間はデータ交換装置を介して情報のやりとりが可能となっている。

H−8811処理装置には，ディスプレイ装置（CRT），XYプロッタ等の出力装置と，ライトペン，ファンクションキー，ダイアル等の入力装置が附加され，CRTに向ったオペレータが全てのグラフィカルなジョブを制御する事が可能となっている。

B　UNGLのハードウェア構成

UNGLは，HITAC−8400処理装置及び周辺装置をハードウェア構成とする。

〔図１－８〕　CGOSにおけるハードウェア構成

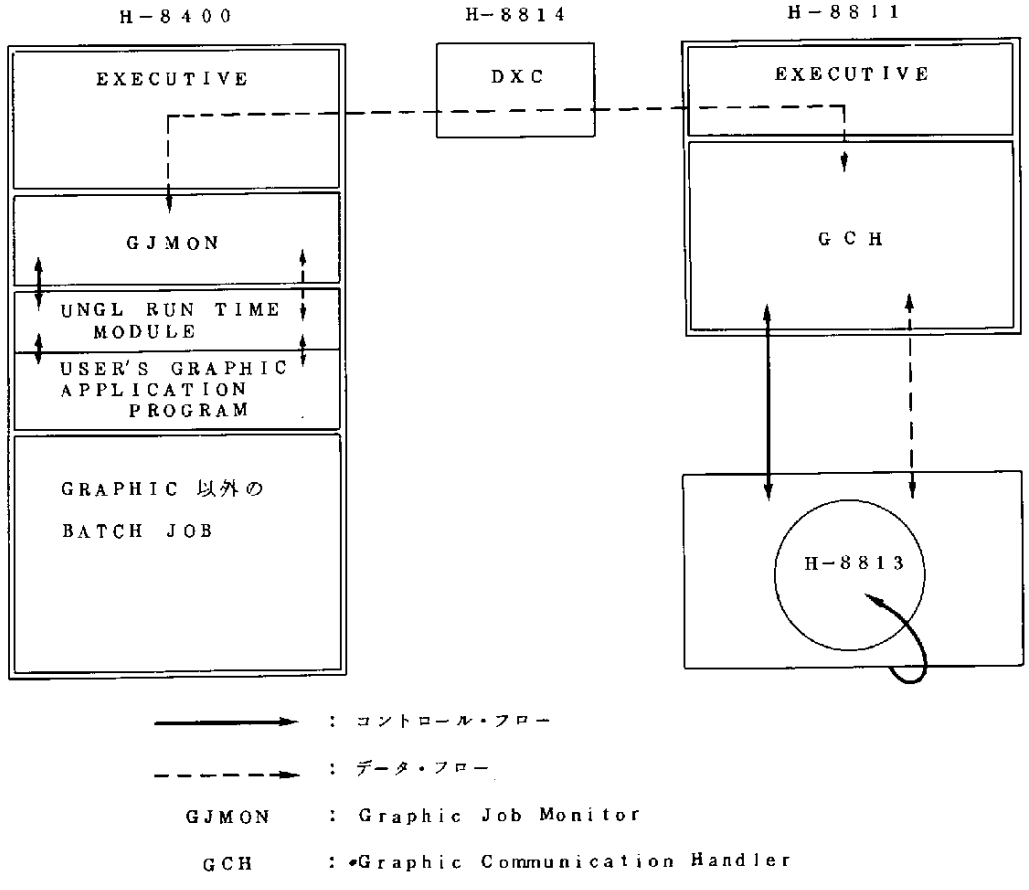


1．3．2　ソフトウェア構成

A　CGOSのソフトウェア構成

CGOSを構成するコントロール・プログラムは〔図1－9〕に示され，H－8400処理装置内のGJMON（Graphic Job Monitor）と，H－8811処理装置内のGCH（Graphic Communication Handler）とからなり，両プロセッサのエグゼクティブと協力して，グラフィック・プログラムの運行を管理する。

〔図1－9〕　CGOSにおけるソフトウェア構成



各コントロール．プログラムの機能概要を次に示す。

a．GJMONは，H−8400エグゼクティブの下で，スレーブモードのタスクとして常駐し，次の機能を果す。

(1) DXC経由によるH−8811プロセッサとの間の相互データ転送。

(2) DXC経由で受取ったメッセージ・データの解析

(3) CRTからのユーザ・プログラムの起動およびモニタリング

(4) グラフィック・コマンドに対する処理

(5) グラフィック・アプリケーション・プログラムとの相互データ転送

b．GCH

GCHは，H−8811エグゼクティブの下に常駐し次の機能を果す。

(1) DXC経由によるH−8400プロセッサとの相互データ転送

(2) システム・レスポンス情報の表示

(3) 図形データから図形コマンドへのコンパイリングおよび図形表示

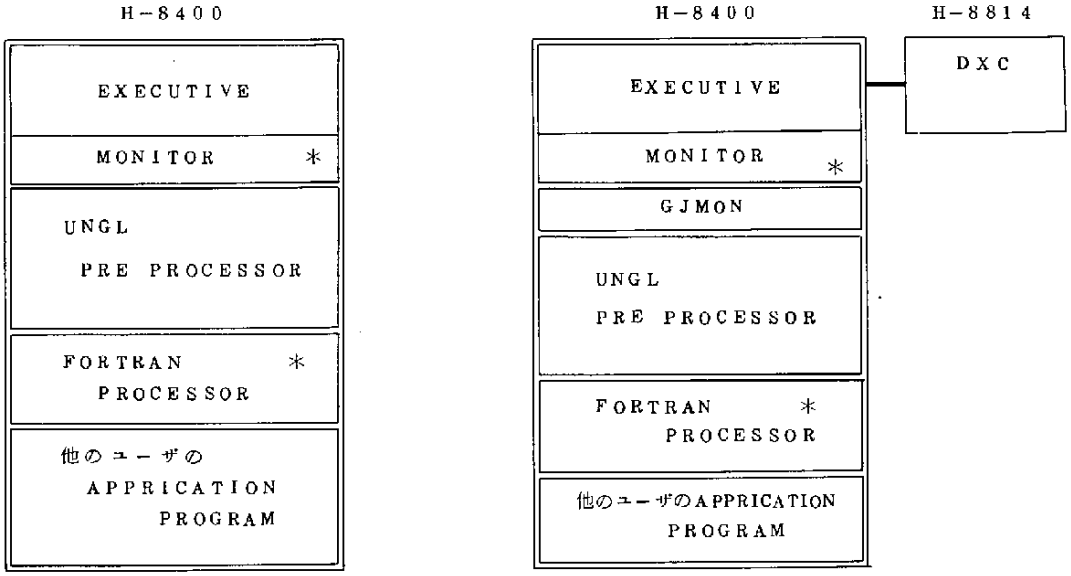
(4) 各種割込み処理

(5) CRT上に表示されている図形のXYプロッタ上へのハードコピー

B UNGLのソフトウェア構成

UNGLを構成するプログラムは，H−8400処理装置内と二次記憶装置内にあり，UNGLプリ・コンパイラは，H−8400エグゼクティブの下でUNGLソース・プログラムの翻訳を行なう。〔図1−10〕にその構成を示す。

〔図１－１０〕　ＵＮＧＬのソフトウェア構成



（ＣＧＯＳを利用しない時）　　（ＣＧＯＳを利用した時）

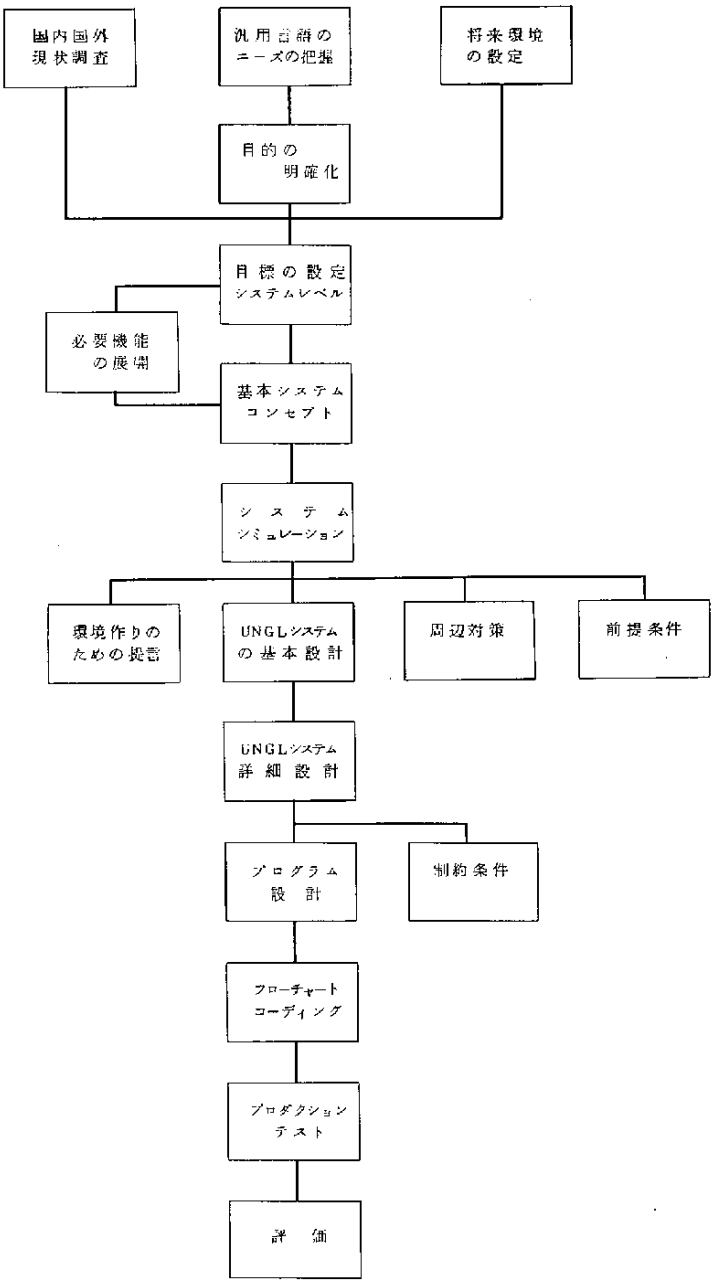
＊印については，ＵＮＧＬプリ，プロセッサがＦＯＲＴＲＡＮプロセッサをローディングした時にのみ存在する。

又，ＵＮＧＬグラフィカル・モジュールは二次記憶装置内にあり，リンケージ・エディタの実行によって，ユーザのグラフィック・アプリケーション・プログラムにリンケージ・バインドされる。

1．4　UNGLの設計基準

UNGLシステムの設計アプローチは〔図1－11〕に示される。

〔図1－11〕　UNGL開発プロセス



汎用グラフィック言語として要求される機能は，①図形データ処理機能，②割込み処理機能，③汎用データ・ストラクチャ操作機能，④非グラフィカル処理機能である。システム設計の基本方針としては，上記4機能をすべてUNGLの中に盛り込む事を考えなければならない。

先ず④の非グラフィカル機能に関しては，ベース言語であるFORTRANのフルセットを活用しているので，FORTRANプログラミングに要する以上の知識は必要としない。ここでFORTRANをベース言語にした理由をいくつか述べる。

(1) FORTRANは現在最も普及している。

(2) 図形モデルの扱いにはFORTRANの算術演算処理がそのまゝ利用できる。

(3) 表現形式がステートメント形式，ファンクション形式を持つ事ができ記述性がよい。

(4) 作成に要する時間が短縮される。

次に①～③のグラフィカル機能に関しては，それぞれが機能別のモジュールとなり，ユーザ・プログラムが必要とする時のみ，ユーザ・プログラムに結合編集される。

この様な機能を満足するシステムの設計にあたって，特に留意した点は次の通りである。

a．汎　用　性

特定のアプリケーション向きの言語でなく，なるべく広範囲のアプリケーション・プログラムが実現できること及び，特定のハードウエア，ソフトウエアの存在を前提とせず，出来る限りマシン・インディペンデントであること。

b．機　能　性

グラフィック・システムの持つ機能を充分に発揮することができ，高度なアプリケーション・プログラムが容易に作成できるような言語である事。

c．信　頼　性

インタラクティブな人間と機械の交信過程において，人間のいかなる誤操作に対してもシステムが適切な対応処置をとり，人間側もシステム側も共に完全なリカバリー（回復）が可能なこと。

d．拡　張　性

ステートメントの種類及び機能に関して将来の拡張性を考慮にいれた言語であること。

e．ハイレベル

グラフィック言語においては，図形データ処理，汎用データ・ストラクチャ操作，割込み処理等一般のプログラミング言語と比べて新しい概念が多い。

これらの複雑な概念を扱い易い適当な形式で表現し得る事が必要である。

以上の様な点を考慮した上で，UNGLシステムの基本設計がなされた。以下にUNGLの持つ性格，プログラム設計上の留意点について述べる。

1．4．1　UNGLの性格

UNGLの性格として備えていなければならない基本的な条件は次の通りである。

a．システムのやりとりが簡単

アプリケーション・ユーザが，グラフィック特有の機能を動作させる場合表現形式が簡単でしかもユーザが動作内容の詳細まで認識しないですむような言語体系をとる。

b．エラーの診断が適切

グラフィカルな機能を実行している途中で発生したユーザの誤りについては，診断ルーチンを動かせるだけでそのジョブをアボートしないような機能を持たせる。

c．結合編集処理

アプリケーション・プログラムが，ファイルのいくつかをリンクして，1つのプログラムとして実行できるような機能をランゲッジに持つ。

d．ファイル処理

ランゲージはソースプログラム・ファイルや，プログラム・ランブラリ，データ・ファイル等の創成，推持の基礎的機能を提供する。又共通ファイルとして多くのユーザが使用可能なファイルを用意したり，ファイルのバックアップ，再成のための機能も提供する。

1．4．2　プログラム設計

プログラム設計上の留意点は次の通りである。

a．システムの全体図の認識

UNGLの機能概要，システムの位置付け，CGOSとの関係などのシステムの全体的な概念を知っておく必要がある。

b．モジュール構造

プログラムの作成，テスト，保守などの面を考慮して各プログラムをモジュール化し，使用頻度の高いモジュールはユーティリティプログラムとして一般化する。

c．プログラミング技法

プログラムの最適化（Optimization）を考え，効率のよいプログラムを作成するために，記憶領域の有効利用，ファイル処理の柔軟性，チェックポイント・リスタートの使用，実行速度の高速化などの点を考慮に入れ，高度なプログラミング技法の採用を図る。

d．コーディング技法

プログラムをコーディングする場合のラベルの統一，コメントの附加などを徹底する。

e．デバッグ体制

診断用ルーチンの組込み，デバッグシステムの作成等とゝもに，コーディング・デバッグの形を明確にし，プログラミングの最適化を図る。

f．文　書　化

プログラム開発の生産性向上のため，設計仕様書，解説書，フローチャート，

保守用ドキュメントなどの文書化を徹底し，標準化も考慮に入れる。

g．プログラミング作業

プログラム開発のための開発計画，開発管理などの作業管理を行ない，定量化された開発状況の把握を行なう。

h．機能間のコミュニケーション

各機能間のインターフェースについては徹底した討論を行ない，必要に応じて共通のコーティリティも作成し，全体が1つのシステムとして効果的に動作するように配慮する。

1．５　ＵＮＧＬの構成

UNGLのプログラムは次の様に分類される。

- プリ・コンパイラ
- グラフィカル処理プログラム
  - 図形データ処理プログラム
  - 汎用データ・ストラクチャ操作プログラム
  - 割込み処理プログラム
  - フレーム処理プログラム

次に各プログラムについてその構成を示す。

1．5．1　プリ・コンパイラ

FORTRANをベースとする翻訳プログラムであり普通のコンパイラのモジュール以外に次の様なモジュールを持つ。

(1)　UNGLステートメント評価モジュール

UNGLステートメントのシンタックス解析モジュールであり，図形データ処理，汎用データ・ストラクチャ処理および割込み処理のシンタックスチェックをおこない，エラーがなければ翻訳しやすい形に直しておく。

(2)　FORTRANソースコード生成モジュール

シンタックス解析の結果作り出された情報をもとに，UNGLステートメントを，FORTRANのステートメントに翻訳を行なう。

(3)　FORTRANコンパイラ呼出しモジュール

ユーザプログラムの全てのプログラム単位が終了した時点で，エラーリストの書出しを行ない，FORTRANコンパイラーの呼び出しを行なう。

(4)　UNGL制御カード解析モジュール

UNGLのパラメータカード，FORTRANのパラメータカード，LINKEDTのパラメータカード等の処理プログラムへの伝達状報を解析する。

1．5．2　グラフィカル処理プログラム

アプリケーション・プログラムの要求に応じて追加されるランタイム．プログラムであり，大きく次のａ～ｄの４つのプログラムに分類する事ができる。

ａ．図形データ処理プログラム

グラフィック・アプリケーションで要求される図形の処理を行なう。図形モデルは，ＣＲＴ上に表現されるグラフィックス特有なもので，それらの構造化および，操作等を扱うモジュールからなる。

(1)　図形モデルの作成モジュール

要求された図形情報から，図形モデルを作成する。この図形モデルは，図形データの構造化されたものであり３次元図形モデル，２次元図形モデルよりなる。図形モデルは，クロス・コネクションを許すトリー構造のデータで，ユーザプログラム領域の下に作成される。

(2)　図形操作モジュール

作成された図形モデルに，回転，縮小，拡大，透視，投影などの変換操作を行なったり，図形モデルの再定義，図形データの変更などを行なう。この変換，変更によって，図形の形状，位置，３次元図形から２次元図形への変換等が行なわれ，ＣＲＴ上に複雑な図形モデルを表示する事が可能となる。

(3)　図形出力モジュール

作成された図形モデルをＣＲＴ上の表示に適したデータ列（濃縮図形データ）に変換する。この時，ＣＲＴ画面と図形モデルの大きさとの関係からウィンドイング（Windowing ）を行なう処理が必要となる。表示図形の動きをダイナミックにするためのムービング指示も行なう事が可能であり，CRT上に表示されている図形の消去，表示情報（輝度，指示可不可）の変更等も同時に行なう事ができる。又，データ列をサテライト側の二次記憶装置にカタログする事を指示したり，二次記憶装置内のデータ列を表示する事を指示する事も可能である。このようにデータ列の作成，作成されたデータ例とＣＲＴの利用に関する情報交換などを処理するモジュールである。

b．汎用データ・ストラクチャ操作プログラム

アプリケーション・プログラムが表現したい問題のモデルは，汎用データ・ストラクチャ操作プログラムによって作成される。問題モデルは，LEAPタイプの連想記憶機構を使用して操作され，関連データ構造（Associative Data Structure）を持っている。この関連データ構造は，問題モデルの表現に非常に適した構造を持っている。この汎用データ処理プログラムは，主に次のようなモジュールで構成される。

(1) 問題モデル作成モジュール

アプリケーション・プログラムの要求によって問題モデルを作成する。関連データ構造は，データ間の関連付けを表現するのに最適であり，同系列データの集合，オーダード・トリプルとして表現される。これらのデータ構造は，ユーザプログラムの表面上には表われず，ユーザは，データ関の相関に関してのみ認識すればよい。関連データは，ホストコンピュータの二次記憶装置に格納される。

(2) 問題モデル操作モジュール

作成された問題モデルの変更，問題モデルに含まれるデータの検索，更新，同系列データの更新，修正などを行なう。特に検索モジュールは，集合演算，DOループによる検索などを持ち，データの効果的な検索を行なっている。又，システムの持つ外部関数によって，FORTRANの実行文の中で，汎用データ・ストラクチャの検索機能を実行する事ができる。

(3) 二次記憶管理モジュール

LEAPタイプの連想記憶機構は，コア内で処理するにはデータ量が多すぎるので，二次記憶装置を用いている。この二次記憶装置の管理を行なうのがこのモジュールである。コア内の内容と，DISK上の内容との関連付け，扱われるデータの読出し，書き込みなどの一連の処理を行なう。特に，初期設定時には，それまでにディスク上に作られた問題モデルの再利用を指定する事もでき，共有ファイルの共同利用の面も管理している。

c．割込み処理プログラム

グラフィック・システムの持つ重要な機能の１つとして，Man-Machine インタラクティブ処理がある。これはグラフィック・ディスプレイを利用するユーザが，計算機と対話する事を要求するために生じたものであり，システムの持つ割込み処理プログラムは，この事を実現するためにユーザのアプリケーション・プログラムに付け加えられる。そのモジュールの内容は次の通りである。

(1) 割込みモデル作成モジュール

インタラクティブ処理をアプリケーション・プログラムの中でスタティックに定義し，割込み処理過程をプログラム状態の遷移という事で表わそうとする時，プログラム遷移表（プログラム・ステートテーブル）を作成する。このプログラム・ステート表は，ユーザの定義する割込みモデルリストであり，単純なテーブル構造になっている。

(2) 割込み要因解析モジュール

ユーザが，グラフィック・ディスプレイの入力装置から割込みをかけると，このモジュールが割込み要因を解析しアプリケーションプログラムで指定されたユーザ割込み処理プログラムへ制御をわたす。又，割込み状態は非同期，同期と２種類のものがあるが，これらの解析，割込み待ちの制御，プログラム間の情報伝達なども行なう。プログラム・ステートが定義されているアプリケーション・プログラムにおいては，プログラム状態の遷移の制御も行なう。

(3) 割込みモデル操作モジュール

割込みモデルをダイナミックに変更したり，許される要因の抑制解除等を行なうモジュールである。割込みモデルの操作は，プログラム状態の遷移，割込み要因の追加，削除，ユーザの割込み処理プログラムの追加，削除などを行なう。

d．フレーム管理プログラム

サテライトコンピュータに常駐し，ホストコンピュータからの指示に従って，図形モデルを二次記憶装置上に書き出したり，二次記憶装置上の図形モデルをＣＲＴに表示したりする。又，二次記憶装置上の図形モデルに修正，変更を行なう。なお，このフレーム管理プログラムは，CGOSのうちのGCHにつけ加えられたルーチンで，サテライトコンピュータに附属する二次記憶装置（ＤＩＳＫ）上の図形モデル（濃縮図形データ）の操作及び，二次記憶装置の管理を行なう。こゝでフレームとは，図形モデルの一画面分に相当し，二次記憶上には，いくつかのフレームが存在する事が可能である。

2. プリ・コンパイラ

2. プリ・コンパイラ

UNGLプリ・コンパイラはUNGLソース・ステートメントを評価してFORTRANソース・ステートメントを作成する。UNGLソース・ステートメントはFORTRANステートメント，UNGLステートメントの2種のステートメントで構成される。

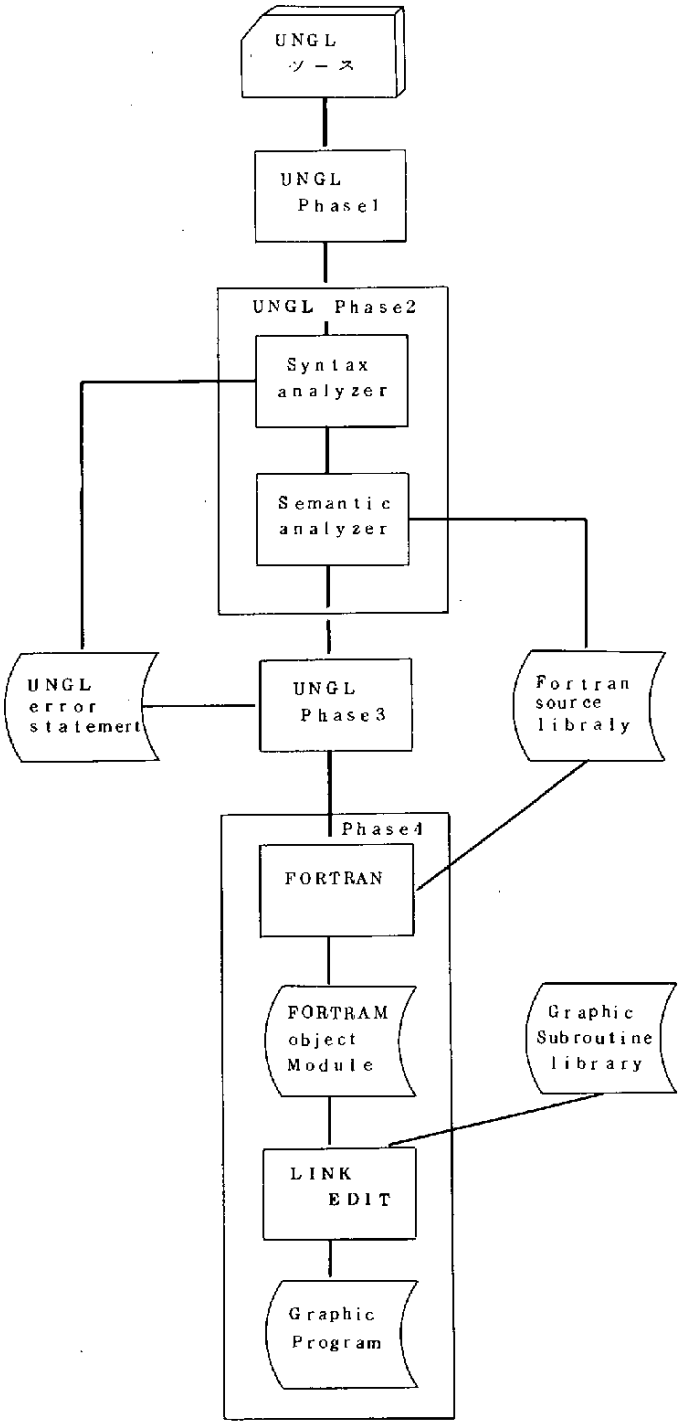
図形データ処理，汎用データ・ストラクチャ操作，割込み処理に関してはUNGLステートメントで記述され，数式処理等に関してはFORTRANステートメントがそのまま使用できる。この2種のステートメントを効果的に記述する事によって，高度なグラフィック・プログラムの作成が可能になる。

以下にUNGLプリ・コンパイラについて記述する。

2．1　概　　要

UNGLプリ・コンパイラは，オブジェクトであるFORTRANソース・ステートメントの作成を主な機能とし，〔図２－１〕に示される処理手順を持つ。

〔図２－１〕　プリ・コンパイラ処理手順



処理手順は大きく４フェーズ（phase）に分類され，１～３フェーズはUNGLの機能であり，フェーズ４は，UNGLとMONITORの機能が組み合わされている。

これらのフェーズの意味及び機能について述べる。

(1) フェーズ１

UNGLフェーズ１は，UNGLプリ・コンパイラに関する制御カードの解析及びタイトル表示用のフェーズであり，UNGLプリ・コンパイラが起動された時，１度だけ実行される。

(2) フェーズ２

UNGLプリ・コンパイラの本来的役割を果すフェーズで，シンタックス・アナライザ（Syntax analyzer）と，セマンティック・アナライザ（Semantic analyzer）に区分できる。すなわち与えられたソース・ステートメントが，文法のルールに従っているかどうかの判定を行ない，従っていれば，翻訳しやすい形に直しておく機能と，シンタックス・アナライザの結果作り出された情報をもとに，翻訳を行ない，オブジェクト・コードを作成する機能を持つ。

(3) フェーズ３

フェーズ３では，UNGLソース・ステートメントのエラーリストの書出しを行なう。すなわちいくつかのプログラム単位が，フェーズ２によって処理された後，ディスク上にスタックされているエラーを，各プログラム単位に編集して書き出す。

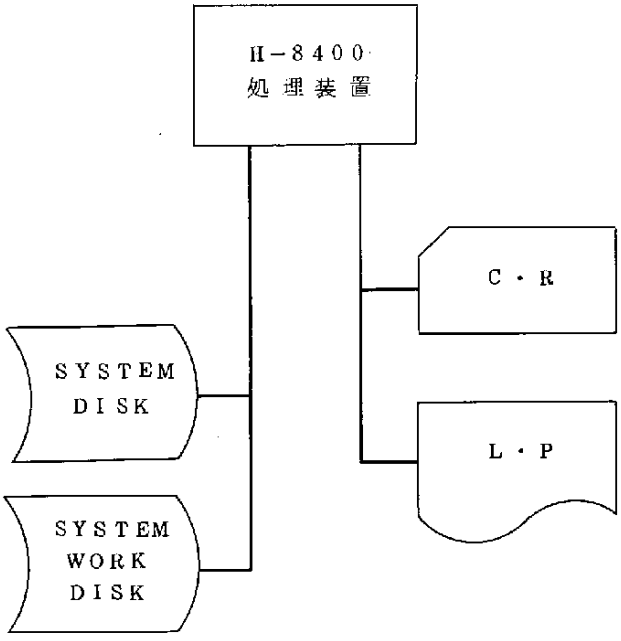
(4) フェーズ４

フェーズ１によって解析されたUNGLプリ・コンパイラに関する制御カードの情報によって，MONITORの起動を行なうか否かを決定し，起動する事になればフェーズ４の処理が行なわれる。なおMONITORの起動は，FORTRANコンパイラの呼び出し，リンケージ・エディタの呼び出し等を意味する。

2．1．1．UNGLの構成

UNGLプリ・コンパイラが動作するために必要なハードウェア構成を〔図２－２〕に示す。

〔図２－２〕　最小ハードウエア構成



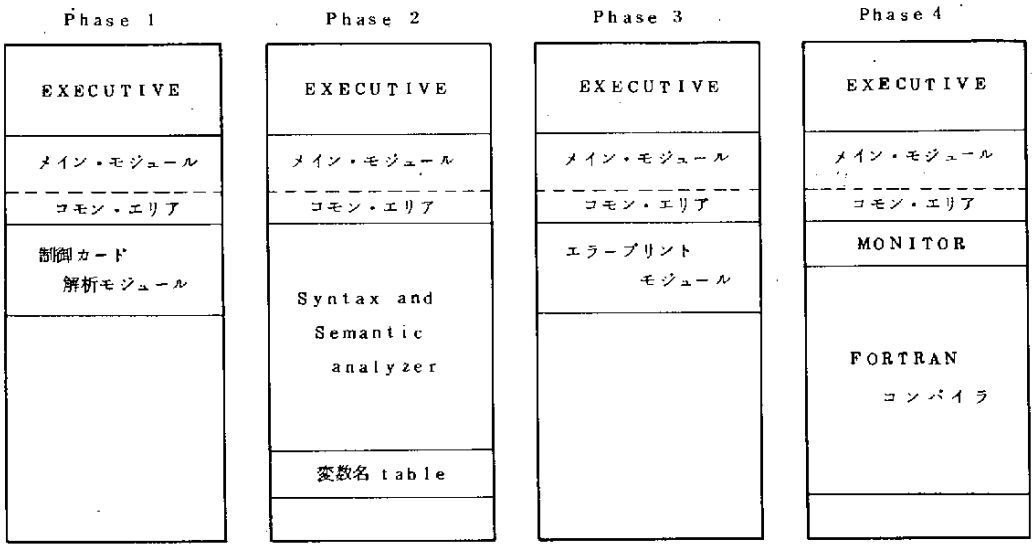
この機器構成は，最小限必要なものであり，その他に，磁気テープの使用も考えられる。又，CGOSを用いてUNGLプリ・コンパイラの起動を行なう事も可能であり，その場合には，CGOSのハードウエア構成が適用される。

次にUNGLプリ・コンパイラのプログラム構成について述べる。

UNGLプリ・コンパイラは，モジュール化された数十種類のルーチンから成り，それぞれのルーチンは，個有な機能を有している。これらは前に述べたフェーズ別にオーバレイされているが，そのオーバレイを制御するメイン・モジュールがルート・セグメントとして主記憶装置上に常駐している。

各フェーズ別の主記憶装置の配置は〔図２－３〕に示される。

〔図２－３〕　phase　別　Coar map

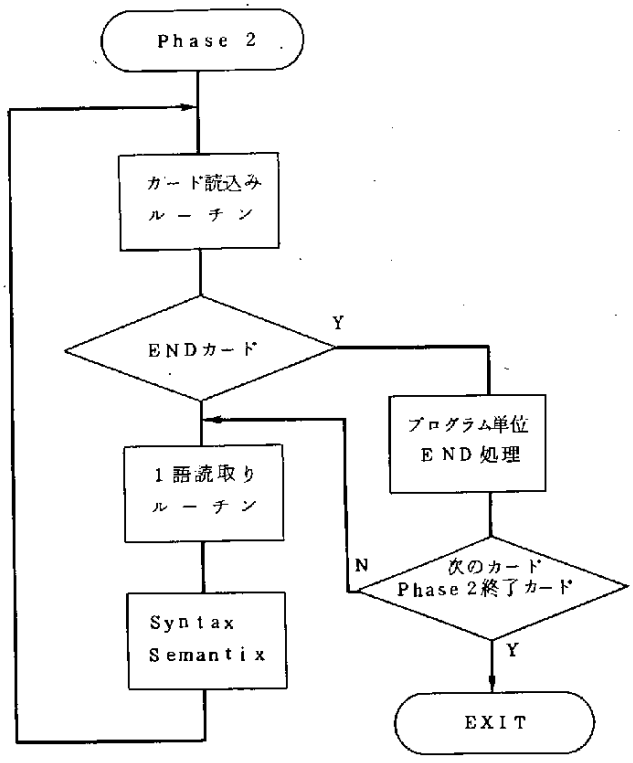


ここで，フェーズ２では特に，UNGLプリ・コンパイラの処理速度を考えて，シンタックス・アナライザとセマンティック・アナライザの全てを主記憶領域に入れるインコアーの形態をとっている。
すなわち，シンタックス・アナライザによって作成された中間コードは，そのほとんどが主記憶装置に持たれ，セマンティック・アナライザによってすぐにFORTRANソース・ステートメントが作成される１パス方式を採用している。このUNGLプリ・コンパイラによって作成されるFORTRANステートメントは，ほとんどがCALLステートメントである。

２．１．２　処理過程

UNGLプリ・コンパイラの処理過程で最も重視される点はｐｈａｓｅ２である。フェーズ２の処理過程は〔図２－４〕に示す通りである。以下にその処理概要を説明する。

〔図２－４〕　phase ２処理過程



１枚のカードがUNGLソース・ステートメントで記述されている場合，カード読み込みルーチンによって，ENDカードがどうかを判定する。ENDカードでなければ，そのステートメントを解析するために１語読取りルーチンによって，区切り記号まで解析し，区切り記号と文字列によって何を表わすステートメントであるかを解読する。UNGLプリ・コンパイラの処理対象となるステートメントであれば，構文解析，翻訳の作業が行なわれ，２次記憶上にFORTRANソース・ステートメントを作成する。その後UNGLソース・ステートメントのリストをプリンタに打出し，次のカードの読込みに移る。

ENDカードの場合には，プログラム単位のEND処理を行なった後，次の

カードを読込む。このカードが，UNGLソース・ステートメントであれば，フェーズ2の処理をくり返す。もしUNGLフェーズ2の終了を示すカード（／／カード又は／＊カード）であれば，フェーズ3へ制御が渡る。

2．1．3　ステートメント処理

UNGLプリ・コンパイラの処理対象となるステートメント及び外部関数は次の様に分類される。

- FORTRANステートメント
  - 非実行ステートメント
  - IFステートメント，STOPステートメント
- UNGLステートメント
  - 定義ステートメント
  - 実行ステートメント
- UNGL外部関係

すなわち，UNGL phase 2では，次の様なステートメント処理用のモジュールを持っていて，その機能は以下の通りである。

(1)　FORTRAN非実行文処理用モジュール

FORTRANの宣言文の解読及び，変数名テーブルへの登録，PROGRAM文，SUBROUTINE文，FUNCTION文，BLOCKDATA文の解読，その他の非実行文の判断及び処理を行なう。

(2)　FORTRAN実行文処理用モジュール

FORTRAN実行文のうち，ＩＦ文，ＳTOP文の判断及び処理を行なう。

(3)　UNGL定義文処理用モジュール

UNGL定義文の解読及び，変数名テーブルへの登録，FORTRANソース・ステートメントの作成。

(4)　UNGL実行文処理用モジュール

UNGL実行文の解読及び，FORTRANソース・ステートメントの作成。

(5) UNGL外部関数処理用モジュール

UNGL外部関数の解読及び，外部関数がFORTRAN規則に合致するように変更。

なお，UNGLステートメント及び外部関数は，ステートメント実行時の機能によって，次の3つの種類に分類する事ができる。

(1) 図形データ処理用

(2) データ・ストラクチャ操作用

(3) 割込み処理用

次にUNGLプリ・コンパイラの設計，各モジュールの詳細について述べる。

２．２　設計方針

UNGLプリ・コンパイラは，UNGLの言語仕様（「Ｓ４５汎用図形処理言語の開発」，及び「２．８」参照）に基づいて記述されたプログラムを解読し，FORTRANのソース・プログラムを出力する。以下に設計基準と開発上の問題点について述べる。

２．２．１　設計基準

UNGLプリ・コンパイラの設計上の前提条件は以下の通りである。

(1)　FORTRANをベース言語とする。

(2)　Object program の Optimization を行なう

(3)　FORTRANコンパイラの起動を行なう。

実際の設計にあたっての問題点は次の通りである。

(1)　言語仕様

FORTRANの構文規則とUNGLの構文規則の相違点を明確にする。

(2)　効率

グラフィック・プログラムは実行時の時間が重視されるので，プリ・コンパイル時間よりもオブジェクトの効率をあげることに重点をおく。

しかしプリ・コンパイル時間も短縮できれば更に望ましいのでイン・コア・コンパイラ方式や，１パス３フェーズ方式を考慮する。

(3)　作成手順

プリ・コンパイラの作成においては，流れ図作成，コーディング，デバッグの各レベル毎に進渉管理体制を強化し，プログラムテストはモジュール単位に行なう。

２．２．２　開発上の問題点

UNGLプリ・コンパイラの設計基準については前に述べたが，さらに下記の項目について開発上の問題点を述べる。

(1)　言語仕様の検討

UNGLステートメントは表現形式の明瞭化を主眼として構成されているが，

次の様な例ではやや問題がある。

PUT　ININ　IN　ININ　　（キー語）

ININ ……セットＩＤ１
ININ……セットＩＤ２

というステートメントでINININと指定されたセットＩＤの中に，ININと指定されたセットＩＤに含まれるメンバーを加えるステートメントではUNGLの記述が空白に意味を持たないので，セットＩＤの区別がつかない。この様な場合には

PUT (ININ) IN　ININ

という事で簡単にセットＩＤの区別がつく。すなわち空白を無視するというFORTRAN規則が前提としてあるので，UNGLステートメントの表現形式を解析可能なように変更した。

(2)　オブジェクト・プログラムの効率化

UNGLで記述された１ステートメントを１つのオブジェクト・ステートメント（FORTRANステートメント）に変換する事は，UNGLプリ・コンパイルの時間を短縮させるが，オブジェクト・プログラムの効率を低下させる。例えば

DRAW　N, IMAGE (X1, Y1, X2, Y2)

というステートメントは，Ｎというフレームに，IMAGEで示されるイメージを表示する事を指定し，その時のウィンドウイングオプション（Ｘ１，Y1, X2, Y2）はそのままの形で渡し，実行時にこれの解析を行なう必要があるが，プリ・コンパイル時にいくつかのオブジェクト・ステートメントに分解しておけば実行時の処理上の手間を省くことができる。

(3)　ルーチン間のインターフェース

UNGLフェーズ２は数十種のルーチンで構成されているが，これらのルーチンは複雑にからみ合って呼ばれる。このためルーチン間の情報の受渡しは重要な問題である。又，あるルーチンが他のルーチンを呼ぶ時には，レジ

スタ類の待避を行なう必要がある。

これらの事を考慮に入れて，レジスタの指定，コモンエリアの使用等に関する一般的規則を持つ。

① レジスタ

各ルーチンに使用されるレジスタについての制限は，〔図２－５〕の通りである。

〔図２－５〕 使用レジスタ

レジスタNo.	意　味
0	使用しない
1	コモンエリア用ベース
2	モジュール用ベース
3	モジュール用ベース
4 〜 13	作業用
14 15	他のルーチンの コール用

② コモンエリアの使用

UNGLプリ・コンパイラの各処理ルーチンが共通に使用する領域としてコモンエリア（common area）を設ける。このエリアにはＩ／Ｏ用のバッファ，各ルーチンのベースアドレス，ルーチン間の連絡に必要な情報の受渡しエリア等があり，ルーチン毎に不必要な領域を持つ事を極力制限している。又，共通領域とする事によって，EXTERNAL，ENTRYなどのアセンブラ命令の使用をなるべくさけるように考慮されている。

(4) テーブル類の検討

UNGLプリ・コンパイラが必要とするテーブルは，大別して次の通りである。

(1) 変数名テーブル

(2) ステートメント・バッファ

(3) エラーメッセージ・テーブル

(4) イメージ代入文用テーブル

(5) オブジェクト・コード用テーブル

(6) モジュール間のインターフェース・テーブル

これらのテーブルについては，各処理モジュールの説明で詳細に示す。

(5) オブジェクト作成

一般にはUNGL 1ステートメントに対して何らかのオブジェクト・ステートメントが出されるが，UNGLプリ・コンパイラがインプリシットに作成するステートメントも存在する。例えば外部関係の型を宣言するステートメントや，データ・ストラクチャ用のファイルオープンを指示するステートメントがそれである。これらのオブジェクト・ステートメントは，プリ・コンパイラによって，必要な場所で出される。

(6) FORTRANソースの見やすさ。

UNGLによって作成されたFORTRANソース・ステートメントを見やすくするためと，デバッグを容易にするためにUNGLステートメントをコメントとして書き出している。例えば

```
DRAW  3, ABC
```

の様なステートメントが解析されると

```
C¥      DRAW  3, ABC
        CALL  S¥WIND (0.0, 0.0, 1024.0, 1024.0, 0, 0,
                      1024, 1024)
        CALL  S¥DRAW (3, 'ABC  ')
```

C ¥

という様にC ¥～C ¥が前後に挿入され且つ元のステートメントが見出しの役目をする。

(7) ＯＳとの関係

UNGLプリ・コンパイラは，フェーズ４でMONITORの起動を行なっているが，これはＨ－８４００ＤＯＳの性格によって制限されるもので，直接FORTRANコンパイラの起動を行なっても，FORTRANコンパイラは動作を行なわない。この事は，マシン・インディペンデントをねらうUNGLプリ・コンパイラの制約条件の１つとなっている。すなわちＯＳの変化によって，FORTRANコンパイラの起動の部分は修正される必要がある。

2．3　UNGLステートメント

UNGLステートメントは大別して定義文と実行文に分けられる。定義文は，FORTRAN宣言文の後で，実行文の前になければならず，UNGL実行文はFORTRAN実行文で許されるいかなる場所にあってもさしつかえない。このUNGLステートメントは，UNGLプリ・コンパイラによって翻訳され，FORTRANのソース・ステートメントに置き換えられる。ここでUNGLステートメントの一般的な表現形式についてFORTRANと異なる点を重点的に記述する。

2．3．1　プログラムの形式

a．UNGL用の文字

プログラム単位は次の文字を用いて書く。

英数字，¥，空白，＝，＋，－，＊，／，（，），，，．，▼，＜，＞，

|，

b．行

行は72個の文字列とし，ホラリスコンスタント，FORMAT文を除いてすべてUNGL用の文字を使わなければならない。又，注釈行，END行，開始行，継続行はFORTRANに準ずる。

c．文

文は開始行のみからなるか，あるいは開始行にいくつかの継続行が続いたものとし，論理IF文の場合を除き，行のうち7から72までのけたに書く。なお，文の番号はFORTRANに準ずる。

d．英字名

記号名（symbolic name）は1～6個迄の英数字からなり，その最初のものは英字でなければならない。

2．3．2　データの型

データの型は，整数型，実数型，倍精度実数型，複素数型，論理型，文字型の6つからなるが，UNGLの定義文のうち，ITEM文，SET文，

IMAGE文については特に文字型のデータを関連づける。

2．3．3　関数の引用

関数の引用は関数名の後にかっこでくくられた関数の実引数を書くが，特にUNGLの定義する外部関数は，データ・ストラクチャ操作用として7つのものが許される。

2．3．4　式

UNGL文で記述される式は要素と演算子によって作られ，使用条件は，データ・ストラクチャ操作に関するUNGLステートメント及び外部関数のうち，セット式の許される個所のみとする。又，演算子は，．＋．，．＊．，．－．の3種類である。

2．3．5　文

UNGL文には実行文と定義文があり，実行文は動作を指定し，定義文はデータの特性及び並び方，プログラム状態などを定義する。

A　実行文

UNGL実行文は，代入文，図形データ処理文，データ・ストラクチャ操作文，割込み処理文よりなる。

a．代入文

代入文にはイメージ代入文，アイテム代入文がある。

(1)　イメージ代入文（image assignment statement）

イメージ代入文はつぎの形とする。

I＜＝e

ここでIは，イメージ定義がされた変数名で，eはイメージ・エクスプレッションである。イメージ・エクスプレッションは，イメージの構造，およびイメージに対する幾何学操作を表わしたものである。詳しくは「3.図形データ処理」の項参照。

(2)　アイテム代入文

アイテム代入文（item assignment statement）は，つぎの形とする。

V←e

ここでVは，アイテム変数定義がされた変数名で，eはアイテム定義がされた変数名又は，配列要素名である。

アイテム，アイテム変数については「５．データ・ストラクチャ操作」の項参照。

b．図形データ処理文

図形データ処理文は，図形データの操作及び図形出力を表わす文よりなり，DRAW文，ANIMATE文，DELETE文，CONTROL文，RESET文，およびDISPLAY文がある。

c．データ・ストラクチャ操作文

データ・ストラクチャ操作文は，問題解析のためのデータ・ストラクチャの創成，操作等を行なう文で，FOREACH文，ALLOCATE文，FREE文，MAKE文，ERASE文，PUT文，REMOVE文，RDATA文，LDATA文，およびIDATA文よりなる。

d．割込み処理文

割込み処理文は，割込み処理に関する記述を行なうもので，ON実行文，TRANSFER文，AWAIT文，ATAKE文，AEXIT文，ON変更文，LOCK文，UNLOCK文，およびSPECIFY文よりなる。

B　定義文

次にUNGLの定義文には，図形データ用定義文，データ・ストラクチャ用定義文，割込み処理用定義文がある。以下にその内容を述べる。

a．図形データ用定義文

図形モデル，サブピクチャーの名称を定義するもので，変数名を文字列として認識する。この定義文はイメージ定義文と称し，図形を表現するための基本的な名称となる。

b．データ・ストラクチャ用定義文

データ・ストラクチャに含まれるアイテム，セット，アイテム変数，ロー

カル変数を定義するもので，アイテム，セット定義文は，変数名，配列要素名を文字列として認識し，アイテム変数，ローカル変数はそれぞれの型をもつ変数として扱われる。

c．割込み処理用定義文

割込み処理の表現としてプログラム・ステートがあるが，このステートを定義するために，次の4つの定義文がある。

(1) DEFINES文

ステート定義の始まりを示す文。

(2) ENDS文

ステート定義の終了を示す文。

(3) STATE文

ステート番号を定義する文。

(4) ON定義文

あるステート番号に含まれる割込み要因，割込み処理ルーチン名，ステート遷移を示す情報などを指定する。

以上のようにUNGL特有のステートメント表現形式が若干あるが，他のほとんどはFORTRANに準拠したものであり，FORTRANの標準的な形式に統一してある。

2．4　機能別モジュール

UNGLプリ・コンパイラのモジュールを使用目的別に分類すると次の8つに分類される。

(1)　メイン・モジュール

(2)　制御カード解析モジュール

(3)　ユーティリティ・モジュール

(4)　FORTRAN非実行文処理モジュール

(5)　FORTRAN実行文処理モジュール

(6)　UNGL定義文処理モジュール

(7)　UNGL実行文処理モジュール

(8)　エラープリント・モジュール

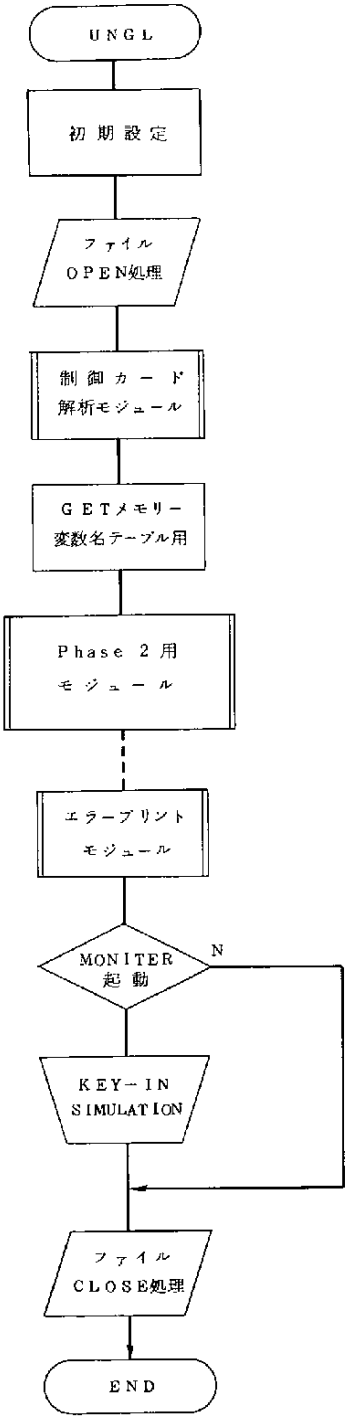
以下に(1)～(5)及び(8)について述べる。(6)(7)については「２．６」「２．７」に詳しく記す。

2．4．1　メイン・モジュール

メイン・モジュールは，UNGLのオーバレイ・コントロール，FORTRANコンパイラの呼び出し，ＥＮＤカード後の処理などを行なう。

このモジュールは，ルート・セグメントとなり，「図２－６」に示す様な処理手順を持つ。

〔図２－６〕　処理ブロック


UNGL
初期設定
ファイル
OPEN処理
制御カード
解析モジュール
GETメモリー
変数名テーブル用
Phase 2 用
モジュール
エラープリント
モジュール
MONITER
起動
N
KEY－IN
SIMULATION
ファイル
CLOSE処理
END

a．初期設定

プログラムの初期化及び，コモンエリアの実データの定義，変数名テーブル等のテーブル類のイニシャライズを行なう。

b．ファイル処理

UNGLプリ・コンパイラで使用するファイル類のOPEN，CLOSE処理を行なう。

c．メモリ管理

ダイナミックなメモリ管理を行なう。すなわち，オーバレイ・コントロール，FORTRANコンパイラ用のメモリ割当て，不要なメモリの解放などを行なう。

d．モニタの起動

KEY－INシミュレーションによって，MONITORの起動を行なう。

e．コントロールカード処理

END処理後のカードの判断を行なう。

2．4．2　制御カード解析モジュール

このモジュールは，UNGLプリ・コンパイラのタイトルのプリント及び次の3つの制御カードの解析，プリントを行なう。

a．＊PARAM

UNGLプリ・コンパイラのための制御情報を指定するもので，「図2－7」にその意味を示す。このパラメータの解析によって得た情報は，コモンエリアに確保され，UNGLのphase 3終了後参照される。

b．//PARAM

FORTRANのためのパラメータを指定するものでこの情報についても，コモンエリアに確保され，FORTRAN実行前にセットされる。この

PARAMカードは，DOSのFORTRAN用のものと同一の表現形式を持つ。

c．OPTION

UNGLのFOREACH文で使用するＤＯの制御変数用の領域を指定するもので，書式は次の通りである。

△OPTION △FOREACH＝××××

××××：４桁の領域長

標準値は５００

FOREACH文で，１番外側のＤＯループに対して中に含まれる制御変数の総和が５００以上になるものについては，このOPTIONを使用する。

〔図２－７〕　＊ＰＡＲＡＭカード

ＰＡＲＡＭ		意　　味
ULIST＝	YES	UNGLソース・ステートメントリストをPRINTに割当てた装置に作成する
	NO	UNGLソース・ステートメントを作成しない。
UMAP＝	YES	MAPをＰＲＩＮＴに割当てた装置に作成する。
	NO	MAPを作成しない。
STACK＝	YES	カード・デック上のソース・ステートメントをＤＩＳＫ上にスタックしてカード・リーダをフリーにする。
	NO	スタック処理を行なわない。
＊1 FORTRAN＝	YES	UNGLコンパイル後にFORTRANを実行する。
	NO	UNGLコンパイルのみで終了する。
＊2 FDISK＝	YES	オブジェクト・コードをＤＩＳＫ上のDOS・SYSU1に作成する。
	NO	オブジェクト・コードをテープ上に作成する。
＊3 ESOURCE＝	YES	FORTRANソース・ステートメントをライブラリ形式にしてFORSSLに作成
	NO	FORTRANソース・ステートメントをFORSSLに作成しない。
＊4 LNKEDT＝	YES	FORTRANコンパイル後にLNKEDTを実行する。
	NO	FORTRANコンパイル後にLNKEDTを実行しない。
＊5 LDISK＝	YES	ロード・モジュールをＤＩＳＫ上のDOS・SYSUT2に作成する。
	NO	ロード・モジュールをテープ上に作成する。

＜注＞＊１　ＹＥＳの時＊２，＊４指定ができ，ＮＯの時＊３の指定ができる。

＊４　は，UNGLソース・ステートメントの後３に　LNKEDT以降のカードを入れる事によって交替できる。

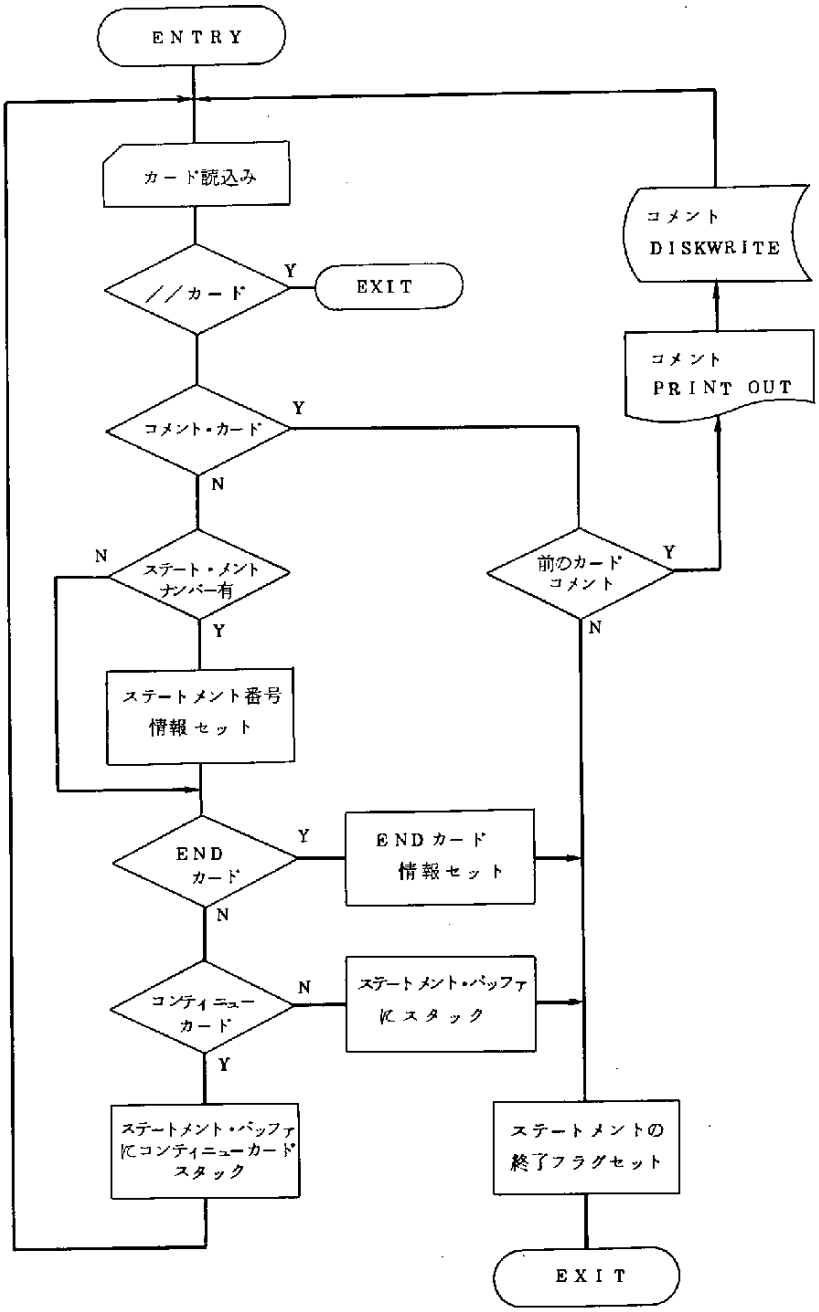
２．４．３　ユーティリティ・モジュール

UNGLプリ・コンパイラに必要とされるユーティリティ・モジュールと内部処理は次の通りである。

a．カード読取りルーチン（CARD　READ）

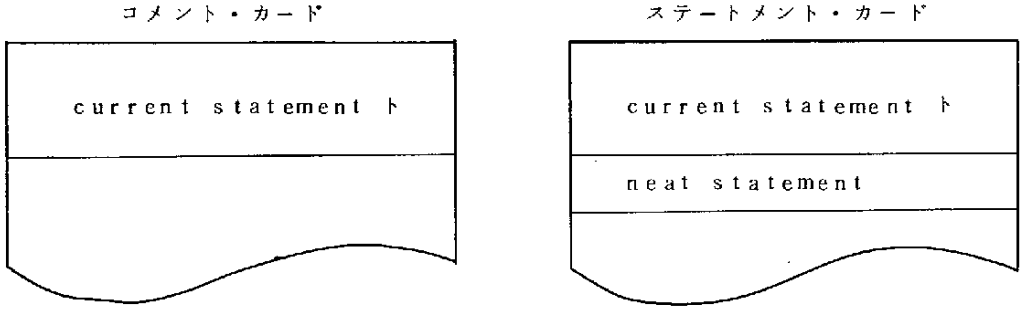
カード１枚を読込み，コメントカード，コンティニューカードの判断及び文番号の評価を行なう。このルーチンではＥＮＤカード，制御カード（//）の解読も重要な機能となっている。〔図２－８〕にその処理過程を示す。

〔図２－８〕　カード読取りモジュール処理過程



このルーチンは，全てのステートメントに対して実行され，コンティニューカードがある場合には，１ステートメントの終りが解析される迄読み続け，ステートメント・バッファに蓄えられる。ステートメント・バッファに蓄えられる状態は，１ステートメントが終了し，次のカードが読取られた状態で決められる。すなわち，先読みを行ないステートメントの終了を判断する。この時先読みされたカードが，コメントカードの場合と，ステートメントカードの場合とでは，ステートメント・バッファの内容が異なる。
〔図２－９〕にその状態を示す。

〔図２－９〕　ステートメント・バッファ



＜注＞　　ト：Statement end mark

この様にカードによって状態を変えたのは，UNGLの処理効率をあげるためである。

次に，カード読取りモジュールでセットされる情報について説明すると，次の通りである。

(1)　現在のカードが，コメントカードかステートメントカードかの判断情報
(2)　ステートメント番号の存在確認情報
(3)　ＥＮＤカード情報
(4)　ステートメント・バッファのティルポインター

このカード読取りルーチンが実行された後，ステートメント解析用のルーチン（実際にはステートメント分枝ルーチン）にコントロールを渡す。

ここでプログラム単位のＥＮＤチェックについて述べる。

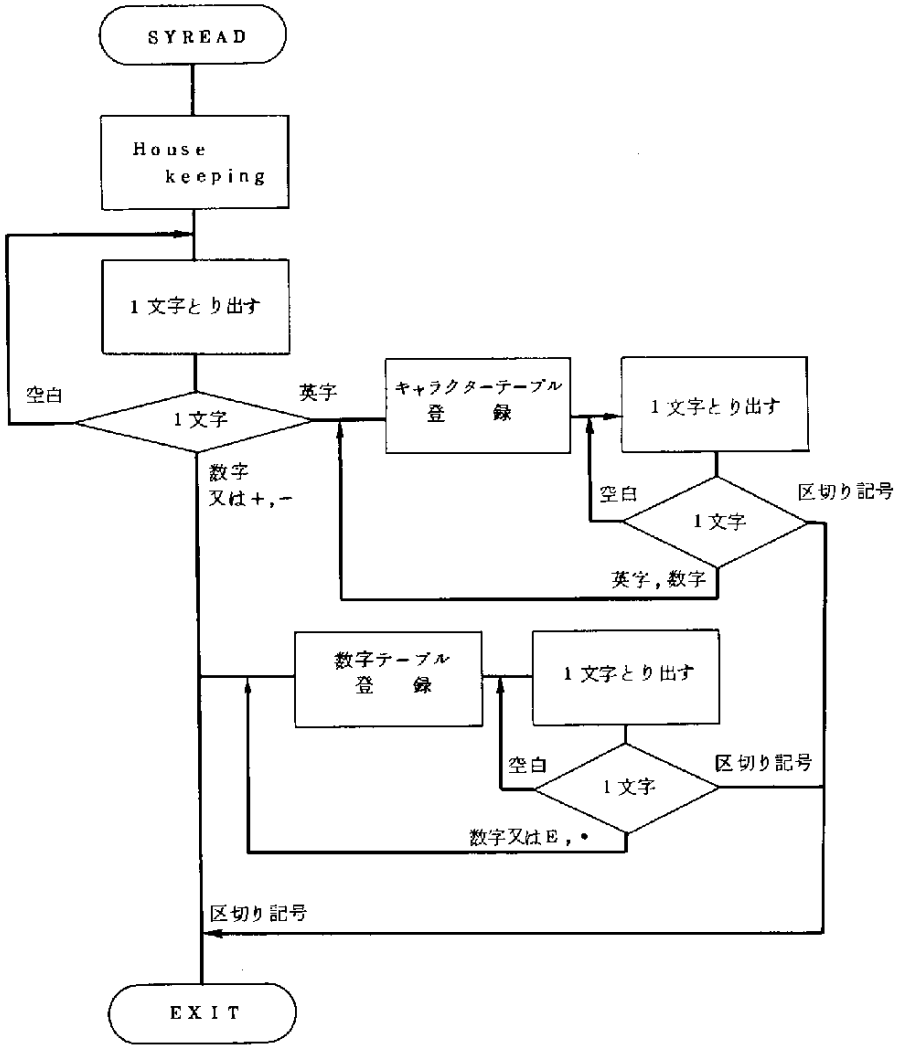
カード読取りルーチンで，ＥＮＤカードの判定がなされると，通常の処理でプログラム単位の終了をチェックできるが，ステートメントの途中でPROGRAM文，SUBROUTINE文，BLOCKDATA文，FUNCTION文があらわれたり，UNGLの終了カード（//ENDUNGL）や制御カード（//，/＊）が読込まれたりすると，ＥＮＤカードが存在しなかったという事になる。

この様な場合には，UNGLプリ・コンパイラは自動的にＥＮＤカードをジェネレートし，ＤＩＳＫ上に書き込みを行なう。又，ＥＮＤ処理後，それまでに作成されたテーブル類の初期化を行なう必要がある。これらの処理が，メイン・モジュールに加えられて実行される。

b．１語読取りルーチン（ＳＹＲＥＡＤ）

ステートメント・バッファに蓄えられたステートメントの１語を読取る。１語の読取りは，区切り符号が見つかるまで，ブランクを無視しながら文字列を読んでいく。この時，数字，英数字，ホラリスコンスタントなどの判断を行なう。〔図２－１０〕にそのアルゴリズムを示す。

〔図２－１０〕　１語読取りモジュールアルゴリズム



又，この１語読取りルーチンは，クォート処理のエントリも含んでおり，クォートマークからクォートマーク迄を読取り，さらに次に表われる区切り記号迄読取る。

なお，ステートメント・バッファの区切り記号の次の文字位置を内部変数

として持ち，カード読取りモジュールが呼ばれた時点で常に初期化される。
又，数字，英数字，ホラリスコンスタントなどの文字数及び区切り記号は
EXIT情報となる。
ここでチェックされる区切り符号は，（ ）＋－・，＜＞＝↑＊／▼ ；の１４
種類である。

c．変数テーブル処理ルーチン（HSRCHR）

FORTRANの変数規則に従う英字名（６文字以内）の検索，登録等を行なう。なおこの変数名テーブルは，ハッシュ技法を使用して検索される。

変数名テーブルには，FORTRAN宣言文，UNGL定義文で評価された変数名が登録され，次の様なテーブル構造を持つ。〔図２－１１〕にテーブルを示す。

〔図２－１１〕　変数名テーブル

Value ID	conf. Flag	Type	array	Ungl Flag

Value ID　：変数名

Conf.Flag　：conflict flag 有▼FF▼ 無▼00▼

Type　：変数型

▼F0▼ INTEGER

▼0F▼ REAL

▼FF▼ LOGICAL

Array　：array check flag 有▼FF▼ 無▼00▼

Ungl Flag　：Ungl 定義文用Flag

▼X1▼ IMAGE

▼X2▼ IMAGE＊2

▼X4▼ IMAGE＊3

▼1X▼ ITEM

▼2X▼ ITEMVAR

▼4X▼ LOCAL

▼8X▼ SET

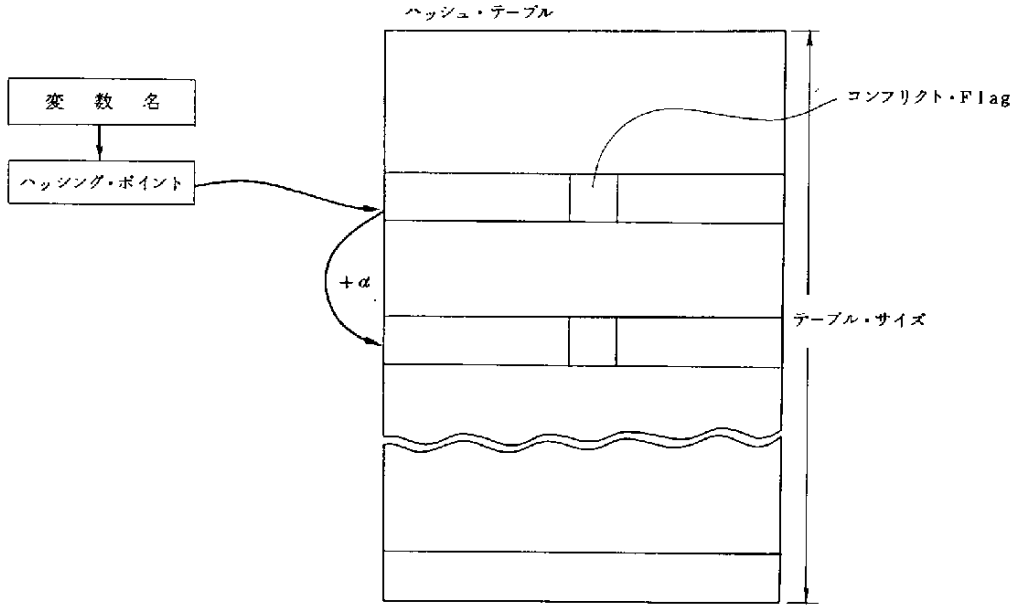
又，変数名テーブルに登録されていない変数のタイプチェック用としてFORTRANの変数規則に従うテーブルも持っている。

次にハッシュ技法について述べる。

ハッシング（hashing）技法は，簡単でしかも効率のよいものを使用している。ハッシングポイントの求め方は

（変数名）／（テーブルサイズに最も近い素数）の余りをリラティブ・ポジションとし，ハッシュテーブルの先頭番地にこのリラティブ・ポジションを加えている。又，コンフリクト（conflict）を起した場合には，ハッシングポイントに素数を加えたものを次のポイントとし，テーブルサイズを上まわる時には，そのポイントからテーブルサイズを減算している。これを〔図２－１２〕に示す。

〔図２－１２〕　ハッシュ技法

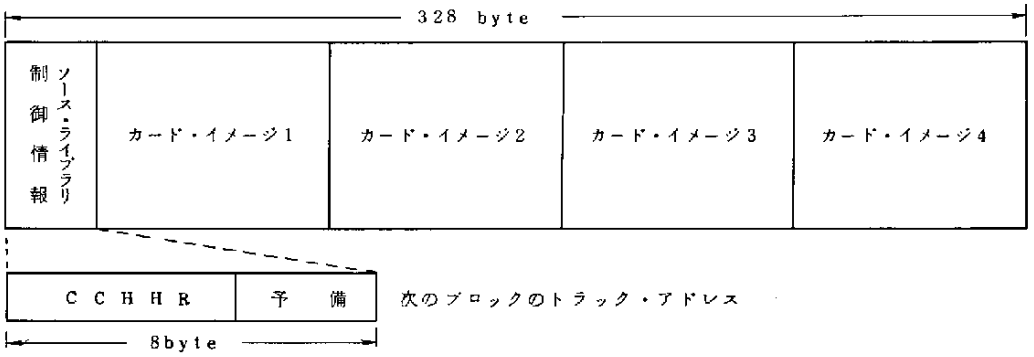


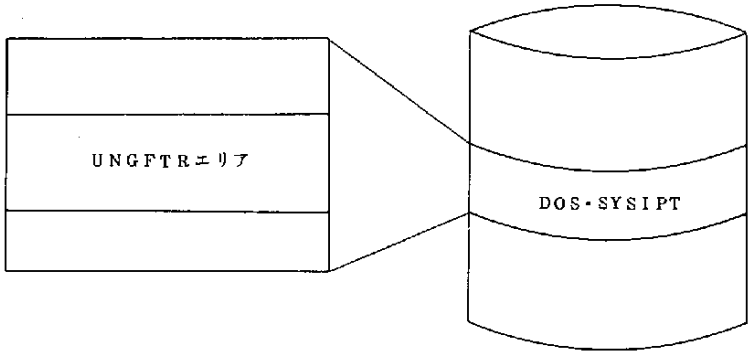
d．FORTRANソース書出しルーチン（DISKWRIT）

FORTRANステートメント又は，UNGLステートメントを評価した後作成されるFORTRANソース・ステートメントは，このルーチンによってディスク上に書き出される。この場合はカードイメージで書き出される。

次にFORTRANソースの書込みエリアと，フォーマットを〔図2－13〕に示す。

〔図2－13〕 FORTRANソース書込みエリアとフォーマット





DISKへの書込みは，ダイレクト・アクセス方式を用い，ディスク上のDOS・SYSIPTで定義されるファイルＩＤのカタログド・プロセデュアエリア（UNGFTR）へ書込みを行なっている。これは，FORTRANコンパイラのローディングに便利なように考慮されている。

e．プリントアウトルーチン（PRINTR）

UNGLのソース・ステートメントの書き出しを行なうルーチンで，1ステートメントの終了毎に実行される。又，コメントの書き出しの際にも利用される。このルーチンでは，プリンタの改ページ等の制御も行なっている。

f．エラー処理ルーチン（ERRPRS，ERDISKP）

FORTRANステートメントのエラー，UNGLステートメントのエラー等を，ディスク上にテーブルとして書く。これを〔図2－14〕に示す。

〔図2－14〕　ERRORテーブル

P.No	error flag	Line Number	E.P

P.No　　：プログラム単位の番号
error flag：UW，UX，UY，UZの分類コード
E.P　　：error massageのpointer

このルーチンは，エラーを評価するルーチンと，エラーをDISK上に書出すルーチンの2つからなりたっている。
又，エラー処理ルーチンの特長として，UNGLソース・ステートメントリストの先頭に，エラーフラグをつけ加えて，エラー個所が見えやすいように考慮している。

ここでUNGLプリ・コンパイラが示すエラーレベルについて以下に述べる。

a．Wエラー

回復可能エラーで，UNGLプリ・コンパイラによって標準値がセットされる。例えば型の違い（2次元イメージと3次元イメージの混合等）はこの表示がなされる。Wレベルのエラーについては，ステートメント解析が引き

続き行なわれる。

b．Xエラー

一部回復可能なエラーで，エラーの生じたステートメントのうちエラーの部分を削除する。その後のステートメント解析は引き続いて行なわれる。このエラーが発生した場合には，コードジェネレートが行なわれる時と行なわれない時とある。

ユーザは，UNGLステートメントの修正を行なわなければならない。

c．Yエラー

回復不能エラーで，エラーの生じたステートメントの全体が削除され，その後のステートメントの解析は行なわれない。

すなわちYエラーの生じたステートメントについては，コード・ジェネレートは行なわれない。このYエラーが発生した場合は，ユーザは必ずUNGLステートメントの修正を行なう事が必要である。

d．Zエラー

UNGLプリ・コンパイラが何らかの原因で処理を停止しなければならなくなった時の表示用エラーで，二次記憶領域のオーバ・フロー，主記憶領域の不足，オーバレイ・ローディングの不可能などによって生じる。

この時UNGLプリ・コンパイラは処理を停止し，ファイルの保護，他の入出力装置の解放等を行なう。

なお，エラーの種類は約２００種類あり，エラーの生じたUNGLステートメントの左にエラーフラグ（UW, UX, UY, UZ）が表示される。

1つのステートメントでエラーレベルが重複する場合には重大なエラーの方が表示される。

g．ステートメント終了処理ルーチン（STMEND）

1ステートメントの評価が終了した時点で，次のステートメント読込みのための準備，プリンタへの書出しの指示，FORTRANステートメントのDISK上への書出しの指示等を行なう。又，UNGLステートメントについ

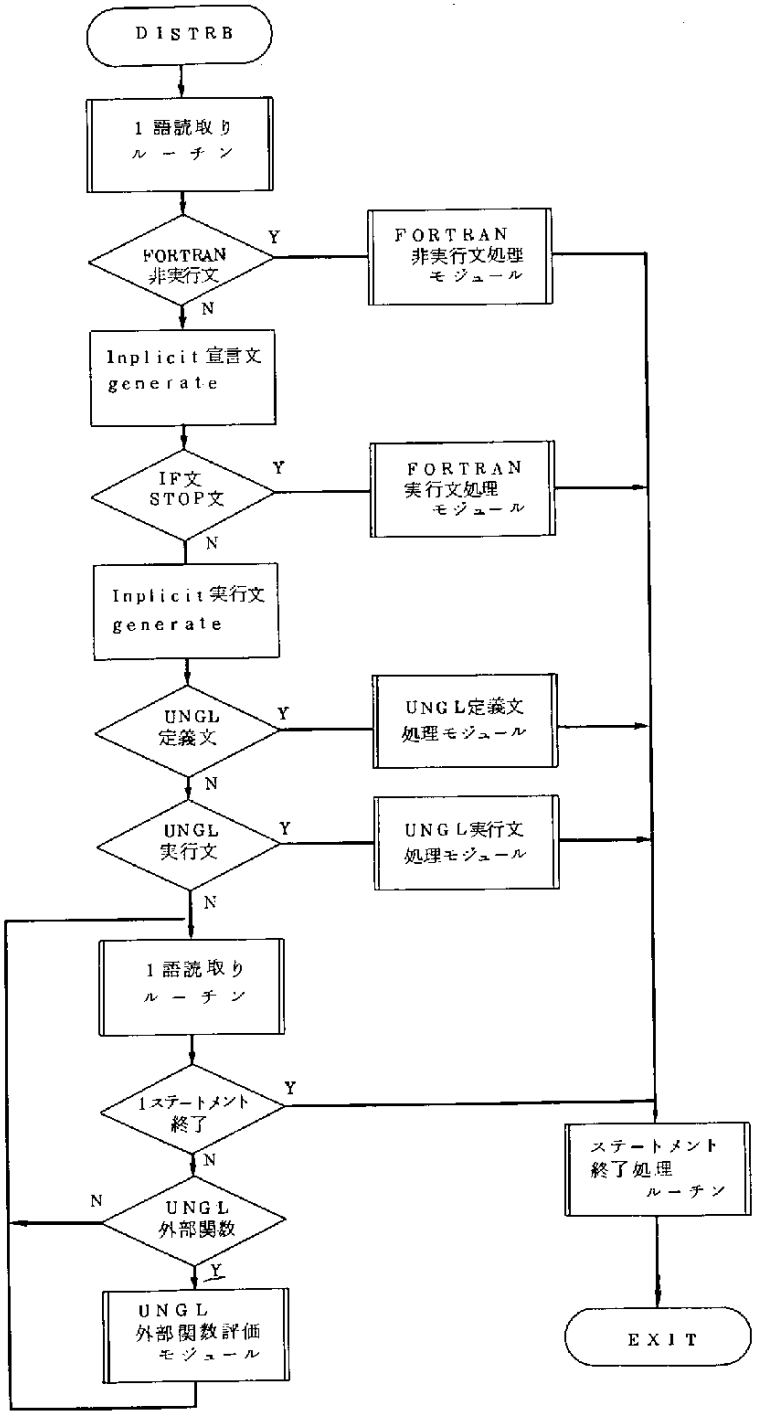
ては，" C¥ "で表わされるUNGLプリ・コンパイラの作成するコメントのDISK上への書き出しも行なう。

h．ステートメント分枝ルーチン（DISTRB）

カード読取りルーチンから制御を渡されステートメントを解読する。すなわちステートメントが実行文か非実行文かの判断及び，UNGLの扱うステートメントの場合には，各ステートメント処理モジュールへの分枝を制御する。各ステートメント処理モジュールのリターン・アドレスは，必ずこのモジュール内にあり，リターン後ステートメント終了モジュールの実行指示を行なう。このルーチンが終ると，カード読取りルーチンに制御を返す。

このルーチンは，各ステートメントの処理を総合的に制御する重要な役割を持つので処理の過程を〔図２－１５〕に示し，そのブロックについて述べる。

〔図２－１５〕　ＤＩＳＴＲＢ処理過程



(1) 1語読取りルーチンの実行指示を出す。

(2) FORTRANの非実行文かどうかを調べ，非実行文であれば，非実行文処理モジュールの実行に移る。

(3) 非実行文でない時には，UNGLプリ・コンパイラがグラフィック・アプリケーション・プログラムに必要とされるインプリシットな宣言文を作成する。但し各プログラム単位に1度ジェネレートされるだけである。

(4) IF文かSTOP文かの判断を行ない，IF文もしくはSTOP文であれば，それらの処理を行なう。

(5) IF文，STOP文以外の時は，(3)と同様な，インプリシットな実行文を作成する。この実行文も，プログラム単位に1度だけジェネレートされる。

(6) UNGL定義文かどうかを調べ，UNGL定義文であれば，UNGL定義文処理モジュールの実行に移る。

(7) UNGL実行文かどうかを調べ，UNGL実行文であれば，UNGL実行文処理モジュールの実行に移る。

(8) 上記以外のステートメントであれば，必ずFORTRANの実行文であるが，このFORTRANの実行文の中で，UNGL外部関数の記述が許されているので，区切り符号"("毎に，UNGL外部関数かどうかを調べる。UNGL外部関数であれば，その評価を行なう。この判断は，1つのステートメントの終了を示す区切り記号迄調べられる。

(9) 各処理モジュールの実行後及び(8)の終了後ステートメント終了処理ルーチンの呼び出しを行なう。

(10) 全ての処理が終ったら，呼出されたカード読取りルーチンに制御をかえす。このうち，(3)に関しては，INPLICIT文，COMMON文，INTEGER文，REAL文，DIMENSION文，EXTERNAL文の作成を行ない，(5)については，データ・ストラクチャ用ファイルの初期化ルーチン，図形データ処理，割込み処理用の領域管理ルーチンのCALL文を作成する。

又，UNGLの記述規則，FORTRANの記述規則により，ユーザはプログラ

ム中でステートメントの記述に際して次の順序で行なう必要がある。

(1) PROGRAM文，SUBROUTINE文，BLOCKDATA文，FUNCTION文などの，プログラムの最初に記述されるべき文

(2) FORTRAN宣言文

(3) UNGL定義文

(4) 上記以外のFORTRAN非実行文，FORTRAN実行文，UNGL実行文

i．括弧処理ルーチン（PAREV）

UNGLステートメントでは，図形データ処理，データ・ストラクチャ操作の記述中に多くの括弧が出てくる。この中には，配列要素名を含んでいる事も多く，解析が複雑であるため括弧処理ルーチンを独立させた。

このルーチンは，左括弧に始まり，対応する右括弧迄の処理を行なう。内部的には，括弧内に不当文字がないかどうかのチェック，括弧の対応がついているかどうかのチェックを行なう。なお，この処理ルーチンは1語読取りルーチンと共用され，括弧の中の文字数，文字列及び対応がとられた右括弧の直後にくる区切り記号が出力情報として出される。

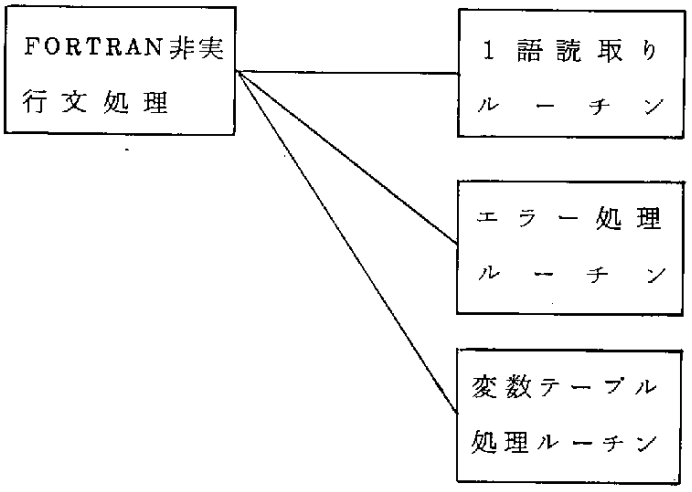
j．引数処理ルーチン（PUTWD）

UNGLのシンタックス・アナライザ後に中間結果として，括弧でくくった文字列，カンマで区切った文字列を出す事があるが，これらの文字列のチェックを行なうためのルーチンで，セマンティック・アナライザによって使用される。

2．4．4 FORTRAN非実行文処理モジュール

FORTRANステートメントの非実行文処理モジュールの構造は〔図2－16〕に示され次のステートメントの解析が行なわれる。

〔図２－１６〕　ＦＯＲＴＲＡＮ非実行文処理モジュール構造



ａ．プログラム，副プログラム文

ＰＲＯＧＲＡＭ文，ＳＵＢＲＯＵＴＬＮＥ文，ＦＵＮＣＴＩＯＮ文，およびＢＬＯＣＫＤＡＴＡ文の解析を行なう。

プログラムの種類の判断，最初の有効なステートメントであるかどうかの判断（ＥＮＤ文の後又は，一番最初の有効なステートメント）等を行なう。

ｂ．宣言文

ＩＭＰＬＩＣＩＴ文，ＩＮＴＥＧＥＲ文，ＲＥＡＬ文，ＣＯＭＰＬＥＸ文，ＬＯＧＩＣＡＬ文，ＣＯＭＭＯＮ文，ＥＱＵＩＶＡＬＥＮＣＥ文，およびＥＸＴＥＲＮＡＬ文についての解析を行なう。

これらのステートメントについては，変数ネーム・テーブルの検索，登録及び，型宣言文，ＩＭＰＬＩＣＩＴ文についてはロケーションチェックを行なう。

ｃ．ＤＡＴＡ文，ＦＯＲＭＡＴ文

ＤＡＴＡ文，ＦＯＲＭＡＴ文については，非実行文である事の判断及びステートメントの表現形式の解析のみを行なう。

２．４．５　FORTRAN実行文処理モジュール

FORTRANステートメントの実行文の評価は，次のステートメントについて行なわれる。

a．ＩＦ文

算術ＩＦ文，論理ＩＦ文の表現形式の解析及び，論理ＩＦ文については，続いて表われるＤＯ文と論理ＩＦ文以外の実行文の解析も行なう。この実行文の中には，UNGLの実行文も含まれる。又，ＩＦ文の中にUNGL外部関数が記述できるので，外部関数処理モジュールの呼び出しを行なう事もある。

b．STOP文

STOP文の前に，汎用データ・ストラクチャ用のファイルのクローズ処理を行なう。

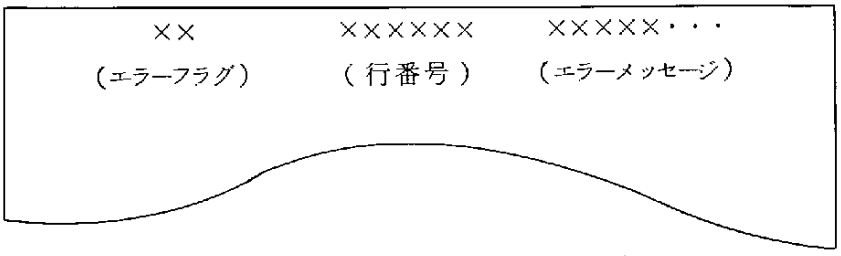
c．他のFORTRAN実行文

上記２文を除いた制御文，代入文，入出力文については，UNGL外部関数の解析とFORTRANの規則に合致するように変換を行なう。FORTRAN実行文のエラーについては，FORTRANコンパイラーの判断に任せる。

２．４．６　エラープリントモジュール

UNGLのエラーは，フェーズ２が終了する迄順次ディスク上に蓄えられている。このモジュールでは，DISK上のエラーに対応する，エラーメッセージを検索し，プリンタに打ち出す。エラー表示用のフォーマットを〔図２－１７〕に示す。

〔図２－１７〕　エラーメッセージ・フォーマット



エラーフラグはUW，UX，UY，UZのいずれかである。

なお，エラーメッセージの検索は，インコアーにおいて行なう。これは，UNGL　プリ・コンパイラの効率を考えたためである。

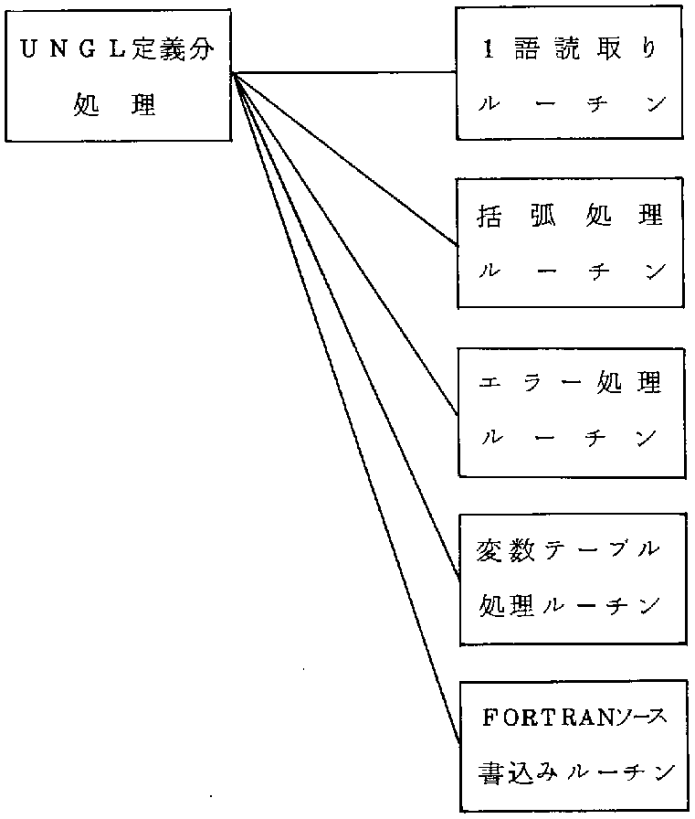
2．5　UNGL定義文処理モジュール

UNGL定義文の処理は，表現形式の解析と，FORTRANのCALL文の作成の2つの処理に分けられる。

UNGL定義文は，UNGL実行文に先だって定義される必要があり，次に述べる2・5・1～2・5・4については，UNGLの変数テーブルに定義した変数名を登録する。

このモジュールの構造を〔図2－18〕に示す。

〔図2－18〕　UNGL定義文処理モジュールの構造



次に各定義文について特に留意すべき点について述べる。

２．５．１　ＩＭＡＧＥ文

２次元イメージか，３次元イメージの指定がされている必要があり，イメージの次元が指定されていない場合には，２次元イメージとみなす。ここで定義されるイメージ名は６文字以内の英字名で，内部的にはグローバルな扱いがなされる。又，１ステートメントで複数個のイメージ名を定義できる。

２．５．２　ＩＴＥＭ文

アイテム名は６文字以内の英字名で，１次元の添字が許される。又添字の中は正整定数のみが許される。さらに／／でくくった中にアイテムの持つデータ長を書く事が許される。このデータ長は正整定数のみである。

又，１ステートメントで複数個のアイテム名を定義できる。

２．５．３　ＳＥＴ文

セット名は６文字以内の英字名で，１ステートメントで複数個のセットを定義できる。

２．５．４　ＩＴＥＭＶＡＲ文，ＬＯＣＡＬ文

アイテム変数名，ローカル変数名は６文字以内の英字名で，定義される変数名は１個以上である。この定義文については，ＦＯＲＴＲＡＮのＣＡＬＬ文の作成は行なわれない。又変数名は定義されているプログラム単位にだけ有効となる。

２．５．５．ＤＥＦＩＮＥＳ文，ＥＮＤＳ文

メインプログラムでのみ使用可能である。すなわち，ステートの定義はメインプログラムでのみ行なわなければならない。又，ＤＥＦＩＮＥＳ文とＥＮＤＳ文は対になって使用しなければならず，その間に許されるステートメントは，ＳＴＡＴＥ文とＯＮ定義文だけである。又，この２つの定義文の２重定義は許されない。

２．５．６　ＳＴＡＴＥ文

ＤＥＦＩＮＥＳ文とＥＮＤＳ文の間でのみ使用可能であり，指定するステートは９９以内の正整定数のみ許される。

２．５．７　ON定義文

ＳＴＡＴＥ文とＳＴＡＴＥ文の間又はＳＴＡＴＥ文とＥＮＤＳ文の間でのみ定義可能であり，指定される割込み要因は，ＬＰＥＮ，ＦＫＥＹ，ＤＩＡＬ，ＴＲＣＭ，ＭＶＥＤ，ＭＥＳＧの６つがＫｅｙ－ｗｏｒｄとして許される。割込み処理ルーチン名は，ユーザが作成する割込み処理用のサブルーチン・サブプログラム名で，ＦＯＲＴＲＡＮの変数規則に従い，ＵＮＧＬプリ・コンパイラによって自動的にＥＸＴＥＲＮＡＬ属性を持つ。又，ステート遷移情報については，ＳＴＡＴＥ文で指定されるステート番号と同様の規則を持つ。ただし遷移ステートについては，ＤＥＦＩＮＥＳ文とＥＮＤＳ文の間で必ず定義されている必要がある。

なお，これらの定義文が，ＵＮＧＬ実行文，ＦＯＲＴＲＡＮ実行文の途中で表われた時には，Ｗエラーが表示される。

２．６　ＵＮＧＬ実行文処理モジュール

UNGL実行文は，図形データ処理用実行文，汎用データ・ストラクチャ操作用実行文，割込み処理用実行文の３つに分類する事ができ，それらは各実行文処理モジュールによって表現形式の解読がなされ，対応するFORTRANのステートメントが作成される。これらの処理に関しては，多くのサブ・モジュールが動作し，オブジェクト効率のよいFORTRANステートメント作成の手段が考えられている。以下に各実行文処理モジュールについて記す。

２．６．１　図形データ処理用実行文

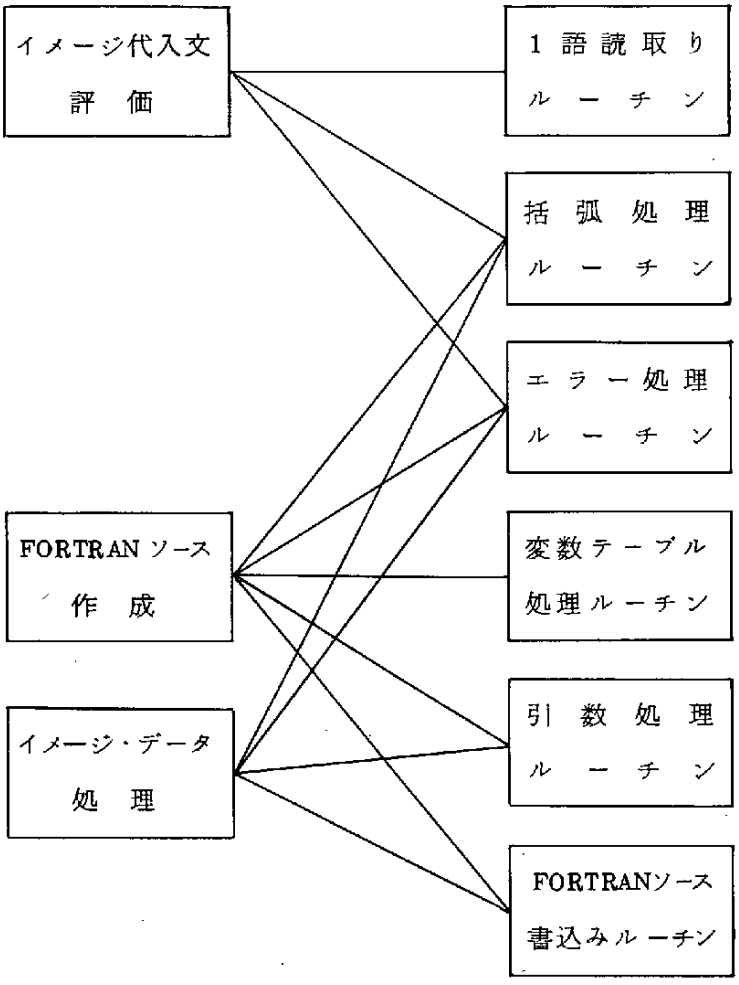
図形データ処理用実行文は，代入文，図形データ処理文に分けられる。これらの実行文は，図形データの作成，構造化，操作，入出力等に関する指定をするために用いられ，ＣＲＴに表示される図形を表現する。詳しくは「３．図形データ処理」の項を参照。

次に代入文，図形データ処理文について示す。

A　代入文

イメージ代入文は，イメージの構造化，実データの定義，変換操作，グルーピングなどを示すエクスプレッションによって表現され，それらの処理結果がイメージに代入される。イメージ代入文処理モジュールの構造は〔図２－１９〕に示される。

〔図２－１９〕　イメージ代入文処理モジュールの構造



イメージ代入文の処理手順は以下の通りである。

(1)　イメージ代入文の表現形式のチェック。

(2)　イメージ代入文をリバース・ポリッシュ・ノーテーション（Reverse Polish　Notation）に変換。

(3)　各オペレータに対するＣＡＬＬ形式のオブジェクト作成。

a．イメージ代入文の表現

イメージ代入文の表現形式の解析は，2次元イメージと3次元イメージの関連性，定義されていないイメージの解釈，変換マトリックス，論理化因子のアーギュメント解析，イメージ・データのアーギュメント解析などである。すなわち

イメージ⇐イメージ・エクスプレッション

の評価である。次にイメージ・エクスプレッションについて述べる。

イメージ・エクスプレッションは，グラフィック・オペレータと，グラフィック・データ（イメージ，イメージ・データ，変換マトリックス，論理化因子）により記述される。グラフィック・オペレータは〔図2－20〕に示すFORTRANの算術演算子のような役割を果たす。

〔図２－２０〕　グラフィック・オペレータ

オペレータ	意　　味	右辺に許される形
⇐	代入操作	イメージ・エクスプレッション
＋	併合操作	イメージ・エクスプレッション
｜	論理的グルーピング操作	論理化因子
.ＴＲＳ２.	２次元イメージの回転，移動，スケーリング操作	座標変換マトリックス
.ＴＲＳ３.	３次元イメージの回転，移動，スケーリング操作	座標変換マトリックス
.ＰＥＲＳ.	透視変換操作	透視変換マトリックス
.ＯＲＴＨ.	平行投影操作	平行投影マトリックス

次にグラフィック・データについて説明する。

① イメージ：ユニークな識別名を持つ変数名

② イメージ・データ：イメージのデータを定義するもので，▼（クォート）と▼でくくられた組であり，その中は，ＬＩＮＥ２，ＬＩＮＥ３，ＴＥＸＴ，ＮＵＭＢＥＲを定義するグラフィック・ファンクションとそのデータによって構成される。

③ 変換マトリックス：　.ＴＲＳ２.，.ＴＲＳ３.，.ＰＥＲＳ.，

．ORTH．の直後に必ず表われかっこでくくられたマトリックス・データである。

④ 論理化因子：

グルーピング操作を表わす｜の直後に必ず表われ，＜＞でくくられた論理データを表わす。すなわちイメージエクスプレッションに表われるグラフィック・オペレータは，⇐ を除いたものであり，カッコでくくる事も許される。それらのオペレータの左辺はイメージ・エクスプレッションであるが右辺には許される型が決っている。

次にイメージ代入文で用いられるデータの表現形式について述べる。

(1) グラフィック・ファンクション用データ

グラフィック・ファンクションの引数として用いられるデータ・タイプを以下に示す。

① LINE2（x_1，y_1，x_2，y_2，x_3，y_3・・・）

2次元のlineを表示するためのグラフィック・ファンクションで（x_1，y_1），（x_2，y_2）・・・を結ぶ折れ線を表わす。なお（x_1，y_1），（x_2，y_2）・・・は各点の座標を表わす。

② LINE2（X，Y，I，J，K）

2次元のlineを表示するためのグラフィック・ファンクションで（X(I)，Y(I)），（X（I＋K），Y（I＋K））・・・

（X（J′），Y（J′））を結ぶ折れ線を表わす。（J－K＜J′≦J）

なお，X，Yは配列名又は配列要素名である。

③ LINE3（x_1，y_1，z_1，x，y_2，z_2，・・・）

3次元のlineを表示するためのグラフィック・ファンクションで（x_1，y_1，z_1），（x_2，y_2，z_2）・・・を結ぶ折れ線を表わす。なお（x_1，y_1，z_1），（x_2，y_2，z_2）・・・は各点の座標を表わす。

④ LINE3（X，Y，Z，I，J，K）

3次元のlineを表示するためのグラフィック・ファンクションで（X(I)，

Y(I)，Z(I))，（X（I＋K），Y（I＋K），Z（I＋K））・・・（X（J'），Y（J'），Z（J'））を結ぶ折れ線を表わす。

なおX，Y，Zは配列名又は配列要素名である。

⑤ TEXT（MOJI，xo，yo，N，T）

（xo，yo）を始点としてキャラクタ・ストリング（MOJIの内容）をN字表示する。Tで大文字か小文字の指定を行なう。なおMOJIはホラリスコンスタント，変数，配列名が許される。

⑥ NUMBER（VALUE，xo，yo，FORM，T）

（xo，yo）を始点としてVALUEの値をFORMで指定されたフォーマットで表示する。Tは大文字，小文字の判別を与える。なおFORMは，Fタイプ，Eタイプ，Iタイプのフォーマットを記述する。

(2) 変換マトリックス・データ

変換マトリックス・データは弧括でくくられ，次の4種のものからなる。

① （xo，yo，θ，Sx，Sy）

2次元イメージの座標変換に用いられ，始点（xo，yo），回転角θ，スケーリングファクタ（Sx，Sy）を与える。

② （xo，yo，zo，θxy，θz，Sx，Sy，Sz）

3次元イメージの座標変換に用いられ，始点（xo，yo，zo），回転角θxy，θz，スケーリングファクタ（Sx，Sy，Sz）を与える。

③ （F，xo，yo，zo）

3次元イメージの透視変換に用いられ，視点（xo，yo，zo），と投象面Fを与える。

④ （F）

3次元イメージの平行投影に用いられ，投象面Fを与える。

(3) 論理化因子のデータ

論理化用の｜オペレータの後で記述され次の様な形式を持つ。

＜GFACT，I＞

GFACTは論理識別名であり，Iは制御情報を表わす。

b．イメージ代入文評価

イメージ代入文はリバース・ポリッシュ・ノーテーションに変換されるが，そのアルゴリズムについて記述する。

先ず演算順序について考えると，イメージ代入文中のグラフィック・オペレータ＋，．TRS2．，．TRS3．，．PERS．，．ORTH．，｜，⇐及び左右の括弧の処理行列は〔図２－２２〕に示される。この演算処理行列より，優先順位関数が〔図２－２３〕に示す様に作成される。

〔図２－２２〕　処理行列

P ＼ C	⊢	)	＋	.TRS2.	.TRS3.	.PERS.	.ORTH.	｜	(
⇐	⋖		⋖	⋖	⋖	⋖	⋖	⋖	⋖
(		≐	⋖	⋖	⋖	⋖	⋖	⋖	⋖
＋	⋗	⋗	⋗	⋖	⋖	⋖	⋖	⋖	⋖
.TRS2.	⋗	⋗	⋗	⋗	⋗	⋗	⋗	⋗	⋗
.TRS3.	⋗	⋗	⋗	⋗	⋗	⋗	⋗	⋗	⋗
.PERS.	⋗	⋗	⋗	⋗	⋗	⋗	⋗	⋗	⋗
.ORTH.	⋗	⋗	⋗	⋗	⋗	⋗	⋗	⋗	⋗
｜	⋗	⋗	⋗	⋗	⋗	⋗	⋗	⋗	⋗

P：Previous operater　　⊢：終了記号

C：Current operater

〔図２－２３〕　優先順位関数

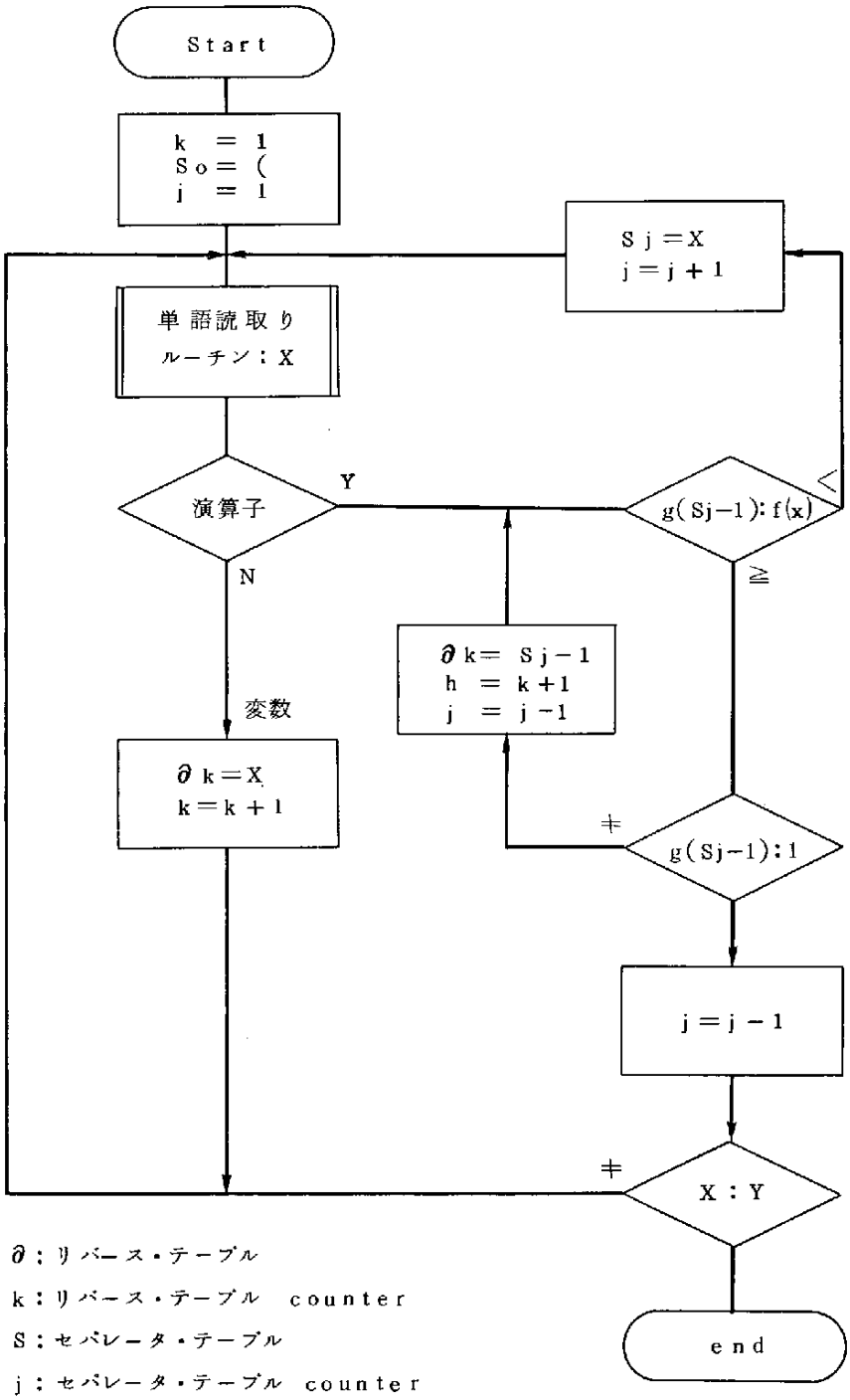
α	g（α）	f（α）
.TRS2. .TRS3. .PERS. .ORTH. |	6	5
＋	4	3
⇐	2	7
（	1	7
）⊢	7	1

g（α）：右のものとの比較関数

f（α）：左のものとの比較関数

イメージ代入文処理については，スタック，優先順位関数が利用されるが，スタック処理のアルゴリズムを〔図２－２４〕に示す。

〔図２－２４〕　イメージ代入文処理



∂：リバース・テーブル

k：リバース・テーブル　counter

S：セパレータ・テーブル

j：セパレータ・テーブル　counter

ここで次の様なイメージ代入文を例として考える。

A⇐（B．TRS2．（X0，Y0，I）＋C）|＜G1＞＋▼LINE2（X1，Y1，X2，Y2）▼|＜G2＞

この表現においてはA，B，Cはそれぞれイメージを表わし，（X0，Y0，I）は座標変換マトリックス，＜G1＞，＜G2＞は論理化因子，▼LINE2（X1，Y1，X2，Y2）▼ は，イメージ・データを表わす。

それが〔図2－24〕に示すアルゴリズムによって解析される順序を〔図2－25〕に示す。

〔図２－２５〕　解析手順

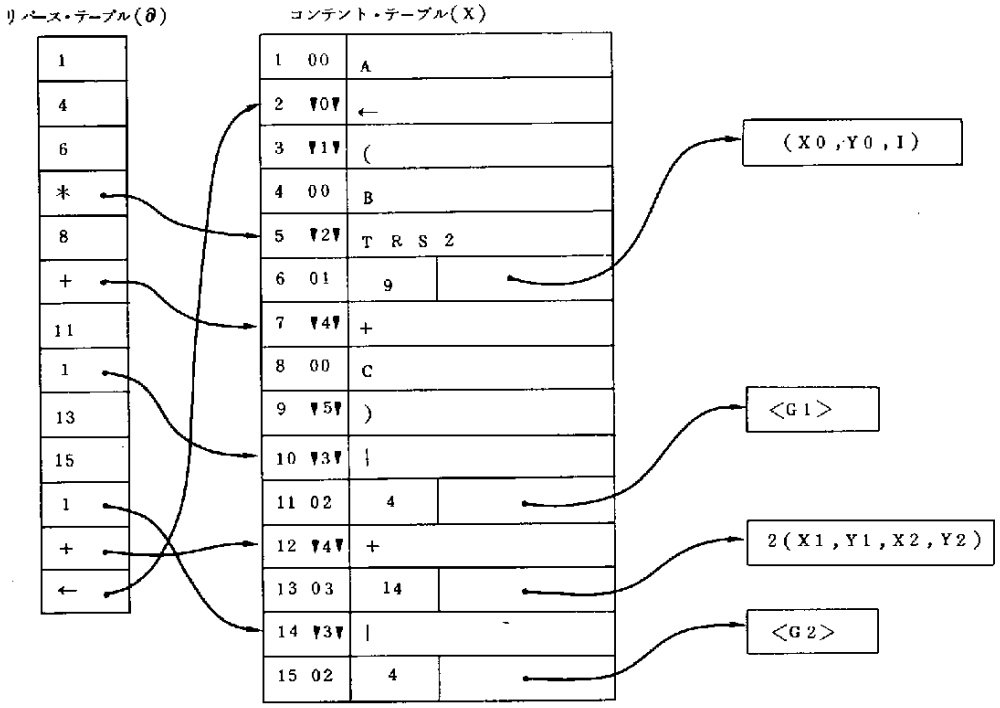
A⇐（B．TRS2．(Xo，Yo，I)＋C)|<G1>＋▼LINE2(X1，Y1，X2，Y2)▼|<G2>

セパレータ・テーブル　リバース・テーブル

(	
	A
(⇐	
(⇐ (	
	A B
(⇐ (＊	
	A B (M)
(⇐ (	A B (M) ＊
	A B (M) ＊ C
(⇐ (＋	
(⇐	A B (M) ＊ C ＋
(⇐ |	
	A B (M) ＊ C ＋ <G1>
(⇐	A B (M) ＊ C ＋ <G1> |
(⇐ ＋	
	A B (M) ＊ C ＋ <G1> | ▼▼
(⇐ ＋ |	
	A B (M) ＊ C ＋ <G1> | ▼▼ <G2>
(⇐ ＋	A B (M) ＊ C ＋ <G1> | ▼▼ <G2> |
(⇐	A B (M) ＊ C ＋ <G1> | ▼▼ <G2> | ＋
(	A B (M) ＊ C ＋ <G1> | ▼▼ <G2> | ＋ ⇐

最終的なリバース・テーブルと，その内容を含むコンテント・テーブルの対応は〔図２－２６〕に示される。

〔図－２６〕　リバース・テーブル



ここで，変換マトリックスデータ，論理化因子，イメージ・データはそれぞれポインターによって，コンテント・テーブルにつなげられている。

すなわち評価された演算子，演算要素がリバース・テーブルに入れられる。そのリバーステーブルをもとにオブジェクト・コードの作成を行なう。

ここで特に注意される点は，＋，⇐ オペレータ以外の オペレータについては，その直後に与えられる演算要素が，データである事と，そのデータが多くのサフィックスを持つ可能性がある事及び，グラフィック・ファンクショ

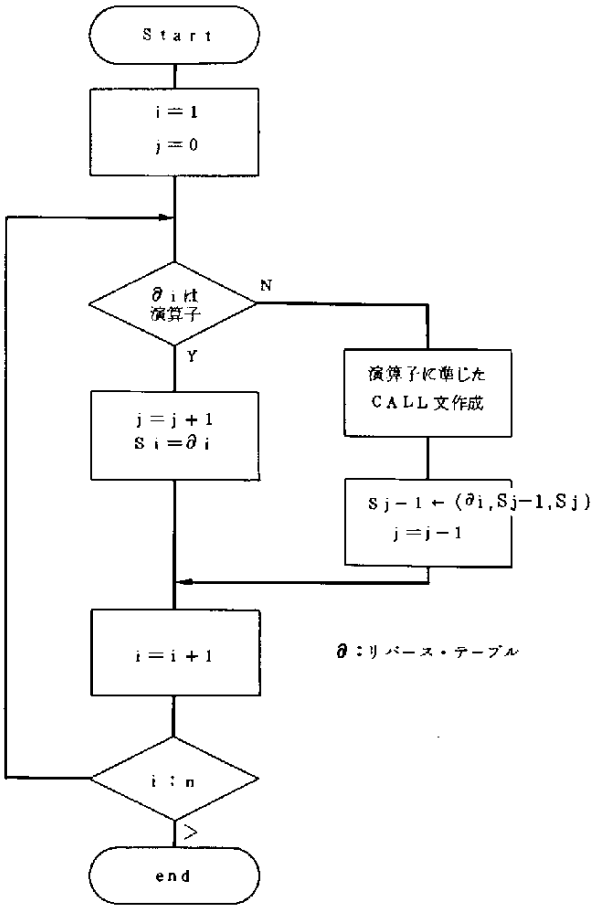
ン用のデータについては特別の処理を必要とする事などであり，これらのデータについてはポインタ指示方式を採用している。

それ以外の処理についてはFORTRANの算術式評価ルーチンに似た方式をとっている。

c．オブジェクト作成

イメージ代入式の評価によって，リバース・テーブルには，演算子，演算要素がスタックされている。このテーブル内は，リバースポリッシュ・ノーテーションの形式である。このノーテーションから各演算子に関するCALL形式のオブジェクトを作成する。このオブジェクトを作成する時の処理を，〔図２－２６〕に示す。

〔図２－２６〕　オブジェクト作成



リバース・テーブルを演算子が見つかる迄検索し，演算子が見つかった時点で直前の２つの演算要素に関して演算を行なう。さらにその演算結果を演算要素の項に入れ，Pop-upの操作を行なう。この時オブジェクトが作成される。すなわちFORTRANのCALL文作成が行なわれる。

各演算子と，それに必要とされるCALL文は「２．８」に示す。

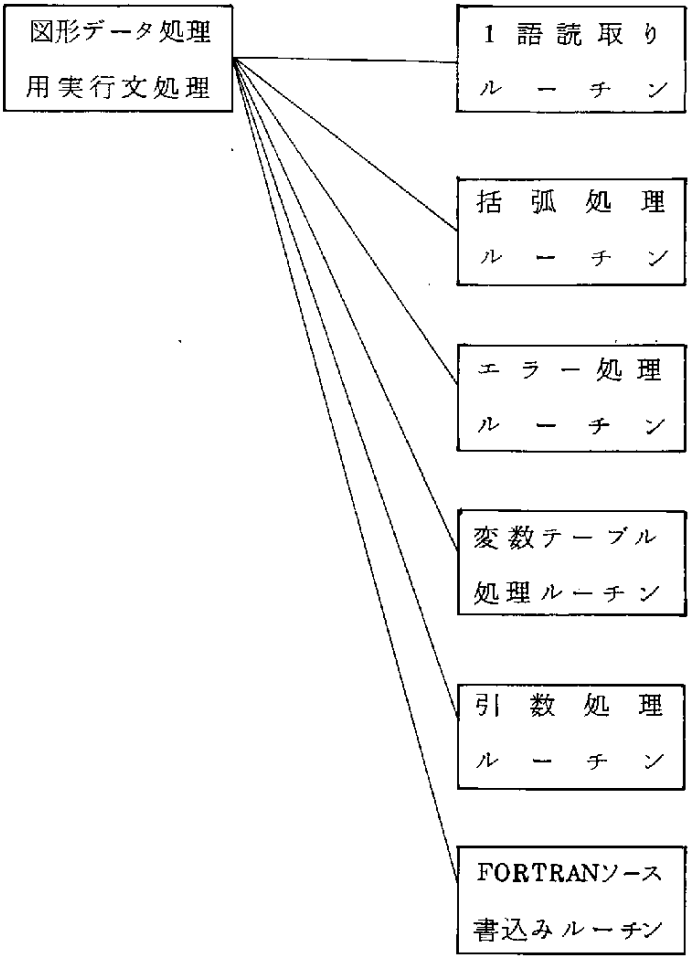
又，イメージ・データのオブジェクトは，各グラフィック・ファンクションについてCALL文を出し，それらのまとまりを最後に指示する。

この様にして，イメージ代入文は３段階にわけて操作され，各操作毎にエラーのチェックが行なわれる。これは第１段階のみでは，ステートメントの完全な評価がなされない為である。

B　図形データ処理文

図形データをＣＲＴへ表示したり操作するためのステートメントからなり，処理モジュールの構造は〔図２－２７〕に示される。これらのステートメントの解析について述べる。

〔図２－２７〕　図形データ処理文の処理モジュールの構造



a．DRAW文

DRAW文には，フレーム番号と出力されるイメージ名称列及びウィンドイング・オプションの記述が行なわれるが，フレーム番号は０～９９迄の正整定数であり，０の時はＣＲＴに直接表示，１～９９の時はＤＩＳＫ上への書出しを意味する。

イメージ名称は，すでに作成されたイメージである事が要求される。又ウィ

ンドイング・オプションは，（x，y，△x，△y，X，Y，△X，△Y）の形式であり，アーギュメント数は4個又は8個であり，実数型，整数型の変数名，定数が許される。又このウィンドイング・オプションは省略が可能であり，省略時には（0.0，0.0，1024.0，1024.0，0，0，1024，1024）の値がとられる。ウィンドイングに関する詳細は「3．図形データ処理」の項を参照のこと。

b．ANIMATE文

この文によって図形の動きを表示する事が可能となる。このステートメントはCRTへの直接表示を意味しイメージ名称列，ムーヴィング情報，ウィンドイング・オプションの記述が許されている。ムービング情報は図形の動きに関するものである。

ムービング情報の詳細は，「45－S002ディスプレイ・システムの研究開発」及び「3．図形データ処理」を参照されたい。又，イメージ名称列，ウィンドイング・オプションについてはDRAW文と同様の扱いを行なう。

c．DELETE文

CRT上に表示されている図形又は，DISK上に書き込まれているフレームの論理化因子で指定されたフラグを持つ図形を消去する。DELETE文には，フレーム番号，論理化因子を記述する。フレーム番号はDRAW文と同様に扱う。DELETE文の実行に先だってDRAW文が実行されている必要がある。又，論理化因子については，イメージ代入文と同様の形式である。

d．CONTROL文

CRT上に表示されている図形又は，ディスク上に書き込まれているフレームの論理化因子で指定されたフラグを持つイメージの図形制御情報を変更する。

CONTROL文には，フレーム番号，および論理化因子を記述するが，これらの扱いはDELETE文と同様である。ただし制御情報は省略できない。

e．RESET文

ＣＲＴ上に表示されている図形又は，ディスク上に書き込まれているフレームを消去する。ここで記述されるフレーム番号についてはDRAW文と同様の扱いである。RESET文の実行以前にDRAW文又はANIMATE文が実行されている必要がある。

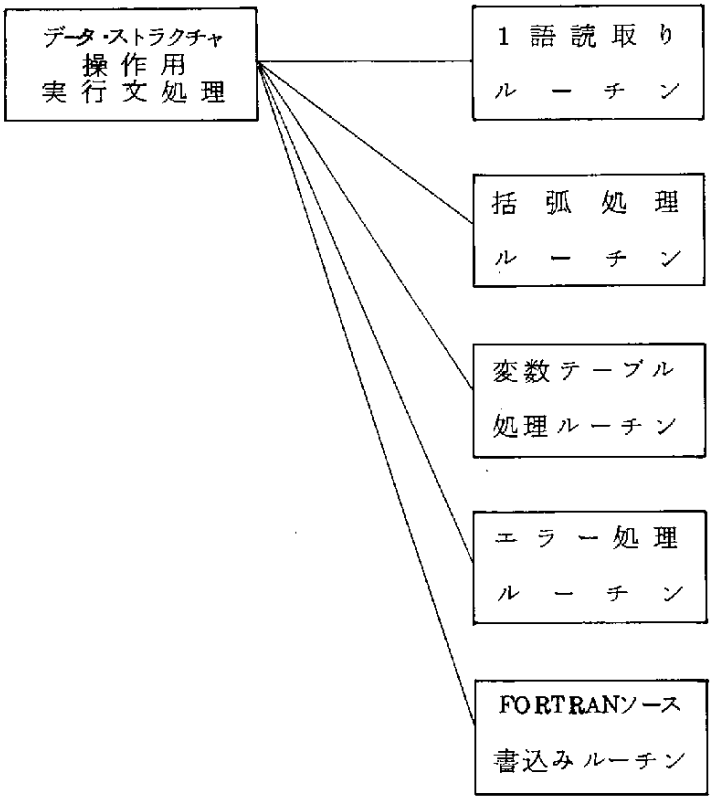
f．DISPLAY文

DISK上に書き込まれているフレームをＣＲＴ上に表示する。ここで記述されるフレーム番号は，１～９９迄の正整定数で，フレーム０を指示しても何ら意味を持たない。

２．６．２　データ・ストラクチャ操作用実行文

データ・ストラクチャ実行文は，代入文，データ・ストラクチャ操作文に分けられる。これらの実行文は，問題解析用データ・ストラクチャの作成，操作に関する指定をするために用いられる。処理モジュール構造を〔図２－２８〕に示す。

〔図２－２８〕　データ・ストラクチャ操作用実行文処理モジュール構造



汎用データ・ストラクチャは，ＬＥＡＰタイプの連想記憶機構を採用した関連データ（Associative Data）によって表現される。詳細は「５．データ・ストラクチャ操作」の項参照。

次にデータ・ストラクチャの表現に用いられるアイテム・エクスプレッションとセット・エクスプレッションについて述べる。

ａ．アイテム・エクスプレッション

アイテムは関連づけの基本要素であり，次のような表現形式をもつ。

(1)　アイテム名称……アイテムのユニークな名称。

(2)　アレイ・アイテム名称……配列をなすアイテムの名称。

(3) アイテム変数名……アイテムの肩代りをする。

(4) ローカル変数名……FOREACH内でのみ使用。

b．セット・エクスプレッション

セットは順序性を持たないアイテムの集合であり，次のような表現形式を持つ。

(1) セット名称……セットのユニークな名称。

(2) セット間の演算子により表現されるセット……．＋．，．－．，．＊．などの演算子によって，セット間の演算がなされたもので例えば，（SETID1．＊．SETID2．＋．SETID3．－．SETID4）

(3) 予約語　ALL……REMOVE文でのみ使用。

以下にデータ・ストラクチャ操作用実行文について述べる。

A　代入文

アイテム代入文は，アイテム変数にアイテム名称のインターナル・ネームをアサインするステートメントで，アイテム宣言によってアイテムテーブルに登録されたインターナル・ネームと対応がとられる。この時アイテム名称に，アレイアイテムを指定する事も可能である。なお，アイテム名は6文字以内の英数字で，内部的に文字列の形式をもつ。

B　データ・ストラクチャ操作文

データ・ストラクチャ操作文は，関連データの創成，変更，検索等の種々の処理を行なっている。

以下にデータ・ストラクチャ操作文の処理について示す。

a．FOREACH文

FOREACH文はある条件を満すアイテムに対して必要な操作を加えられる様な表現形式を持ち，FORTRANのDO文と同様な処理を含んでいて，ネスト表現が可能である。先ず表現形式について説明する。

(1) 表現形式

① FOREACHローカル変数　INセット・エクスプレッション　DOn

② FOREACH　context・spec　DO　n

n：端末文のステートメント番号

①は，セット・エクスプレッションで評価されるセットの構成要素を１つづつローカル変数に割当て，１回づつすべてのメンバーについてプロセデュアを実行する。

②は，オーダード・トリプルの１つ又は２つの構成要素が，ローカル変数に割当てられ，その集合に対してプロセデュアを１回づつ実行する。

(2) オブジェクト作成

オブジェクトの作成は，CALL文と，DO文に分けられる。CALL文については，表現形式に従って作成されるが，DO文に関しては，特に制御変数の扱いに注意を要する。制御変数はFOREACH文のネストに入る毎にインクリメントし，端末文に出合うとディクリメントする。FOREACH文のオブジェクト例を示す。

```
      FOREACH   (ITEM1, ITEM2, X)  DO   100
        :
100   CONTINUE
```

前記のステートメントにより

```
      CALL    H¥ESTM(is, ie, iss, o, triple)
      DO   100      ¥i=is, ie
        :
100   CONTINUE
```

(3) ローカル変数の扱い。

ローカル変数は，データ・ストラクチャ操作の処理を記述する際にFOREACH文のループの中でのみ有効であり，FOREACH文で定義される毎にループレベルを与えられる。定義されたローカル変数は，内部変数¥A（¥i）との対応テーブルに登録され，自分を含めた下位のループに含まれるローカル変数は全て¥A（¥i）に変換され，アイテム変数の扱いを

受ける。又端末文に出合うと対応テーブルより取り除く。ローカル変数を含む事のできるステートメントは，FOREACH文，IDATA文，RDATA文，LDATA文，REMOVE文，PUT文，MAKE文，FREE文，アイテム代入文及び，ISMBER関数，ISTRPL関数である。

b．ALLOCATE文

配列名で指定された配列に格納されている頭6文字をアイテム名称とし，データ長で指定されたデータの長さを持つアイテムを動的に作成する。その時のアイテムのインターナル・ネームがアイテム変数にアサインされる。

c．FREE文

アイテム名称列で指定されたアイテム又はアレイアイテムを消去する。このアイテムが存在しない時には無視される。

d．MAKE文

かっこでくくられたアイテム又はアレイアイテムの三つ組を，オーダード・トリプルとして創成する。もしこのトリプルの要素となるアイテムが定義されてない時はこのステートメントの実行に先だってアロケートされる。又，アイテムの代りにアイテム変数を使う事ができる。

e．ERASE文

かっこでくくられたアイテム又は，アレイアイテムの三つ組をオーダード・トリプルから除去する。このトリプルが存在しない時は無視される。この場合にもアイテムの代りにアイテム変数を使う事ができる。

f．PUT文

IN以降に表現されるセット名称をもつセットのメンバーに，アイテム，エクスプレッション・リストで指定されたアイテム，又はセット・エクスプレッションで指定されたセットのメンバの全てのアイテムを加える。

g．REMOVE文

FROM以降に表現されるセット名称をもつセットのメンバーから，アイテム，エクスプレッションリストで指定されたアイテム，又はセット・エク

スプレッションで指定されたセットのメンバの全てのアイテムを取り除く。この時予約語ＡＬＬの使用によって，指定されたセットの全てのメンバを取り除く事もできる。

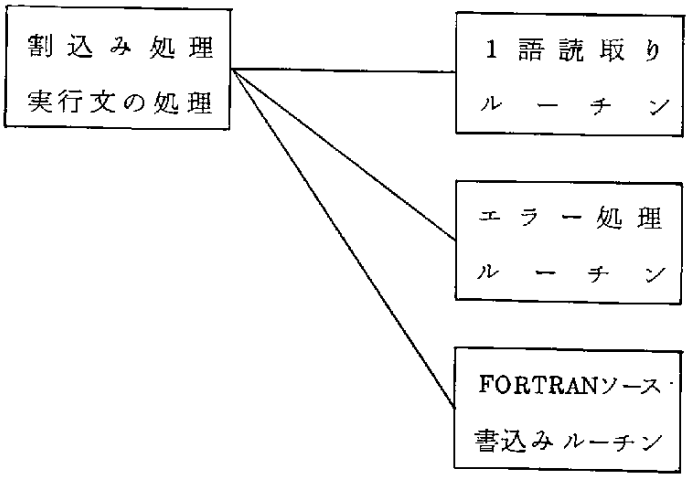
h．IDATA文，RDATA文，LDATA文

指定されたアイテムの指定されたセル番号のところに，＝以降に示される算術式の結果，変数，定数を格納する。この時左辺の型と右辺の型が一致することが必要である。なお右辺には，FORTRANの算術式，論理式，関係式，UNGLの外部関数が記述される。

２．６．３　割込み処理用実行文

割込み処理用ステートメントは，インタラクティブ・プロセスの理解を助け，割込みの概念，プログラム・ステートの概念を明確化している。割込み処理においてはプログラムの状態，遷移が非常に重要な概念となるが，一般のFORTRANユーザは，割込みという問題に対する感覚が薄いので，それを補助するためのステートメントが必要である。UNGLでは割込み処理過程の記術，操作，プログラム状態，遷移という事項を簡単な表現形式で表わせるようにしている。このステートメントに対する処理モジュールの構造を〔図２－２９〕に示す。

〔図２－２９〕　割込み処理用実行文の処理モジュール構造



以下に実行文の表現形式と処理について述べる。

a．ＯＮ実行文

プログラム・ステートを意識せずにＰＬ／Ｉ等にみられるＯＮ文による割込みを考える時用いられ，割込み要因，割込み処理プログラム名を記述する。割込み要因，割込み処理プログラムに関してはＯＮ定義文の記述と同様である。

b．ＴＲＡＮＳＦＥＲ文

プログラム・ステートの遷移を動的に変更する文でステート番号を記述するが，ステート定義が記述されない場合は，実行する事ができない。このＴＲＡＮＳＦＥＲ文は，割込み処理プログラム以外のプログラムに記述されると実行後すぐにプログラム状態の遷移が行なわれるが，割込み処理プログラムで記述されると，その割込み処理プログラムが終了した時点でプログラム状態の遷移がなされる。又ステート定義が行なわれていても，指定するステートが見つからない場合には，このステートメントは無視される。

c．AWAIT文

プログラムを割込み待ちの状態におくステートメントで，一般に割込み処理プログラム以外で使用される。

AWAIT文の記述は，以前にON実行文が実行されているか，TRANSFER文が実行されているかのどちらかの状態でないと意味がない。

これらの実行文が実行されてないと割込み待ち状態にあっても何らの割込み処理も行なわれず，プログラムがシステムループのまま進行しなくなる。

d．ATAKE文

プログラムを割込み待ちの状態におき，ATAKE文に記述された割込み要因の解析のみを行なう。このステートメントの記述方式は

ATAKE　　n，割込み要因〔，割込み要因・・・〕

であり，CRTからの何らかのアクションが，指定された割込み要因の何番目のものであるかを調べ，nにその番号が入る。n＝0の時は，指定された割込み要因以外のアクションである。これらの判断の後，システムループから脱出する。

このステートメントを使用する事によって，プログラム・ステートを一時的に無視した割込み処理を行なう事ができる。

ATAKE文は，割込み処理プログラム内でもそれ以外でも記述できるが，特に割込みプログラム内での使用は，多重ウエイトの性格を持つ。

e．AEXIT文

プログラムの割込み待ちの状態を脱出するためのステートメントであり，割込み処理プログラムでAEXIT文を記述しないと，プログラムはRETURN文によって再び割込み待ちのシステムループに存在する事になる。

f．ON変更文

プログラム・ステートを意識するユーザが，1度定義したプログラム・ステートの内容を変更しようとする時記述され，以前にステート定義の実行がなされていない場合には意味を持たない。表現形式は，ON定義文の形式の

あとに変更するステート番号を指定したものである。

このステートメントは，割込み要因，割込み処理プログラム名，ステート遷移指定の記述ができ，割込み処理プログラム内でも割込み処理プログラム以外でも実行可能である。但し割込み処理プログラム内で実行した場合には，そのプログラムから復帰した時，変更されたステートの内容が有効となる。又，ステート定義を行なうプログラム内では，事実上意味を持たない。

g．LOCK文，UNLOCK文

これ等のステートメントは，割込み要因のロック，アンロックを記述するもので，割込み抑制とその解除のために用いられる。表現形式は，抑制又は解除が行なわれる割込み要因を記述するが，その後にプログラム・ステート番号を書く事も許される。ステート番号を指定した場合には，ステート番号で指定される割込み要因のロック，アンロックがなされる。ステート番号の指定がない場合には，これ等のステートメントが実行された時の受付け可能な割込み要因に対して行なわれる。すなわち，割込み要因がそれまでにＯＮ実行文，ＴＲＡＮＳＦＥＲ文によって受付け可能になっている事が必要である。

h．ＳＰＥＣＩＦＹ文

割込みモデル用の入力文としてＳＰＥＣＩＦＹ文がある。これは，何らかのアクションがあった時に，そのアクションの種類，そのアクションに特有な情報を得るもので詳しくは「４．割込み処理」の項を参照されたい。

割込み情報の詳細な内容を知るＳＰＥＣＩＦＹ文は，配列名又は配列要素名を記述する。この配列については，ユーザが任意に定義してよいが，割込み情報の内容によっては，配列の扱いに注意する事が必要である。

このステートメントは，１度実行されると，割込み情報を必ず配列に格納するが，２度，３度と配列名の異ったものを実行すると，１番近い過去に実行された配列名が有効となる。割込み処理プログラムが多数存在するアプリケーション・プログラムにおいては，格納しようとする配列に対して，ＣＯＭＭＯＮ宣言されている事が望ましい。

２．７　外部関数

UNGLには，アイテム，セットに対して適用される各種の外部関数が用意されている。外部関数は，FORTRANの算術式又は論理式中で１次子として引用される。引数の並びに書かれている実引数は，対応する仮引数と順序，型が一致しなければならない。外部関数の引用の中の実引数は，つぎのいずれかである。

(1)　アイテム・エクスプレッション

(2)　セット・エクスプレッション

(3)　変数名

外部関数が引用される時，UNGLプリ・コンパイラは，実引数についての評価を行なう。具体的にはアイテム・エクスプレッション，セット・エクスプレッションについて，FORTRANの実行可能な形式に変換を行なう。この変換アルゴリズムを次に示す。

外部関数はFORTRANの算述式，論理式中に頻繁に表われるが，ISNMBRを例にとって考えると

IEXAM＝ISUM＋ISNMBR（ＳＥＴ１）＊１０

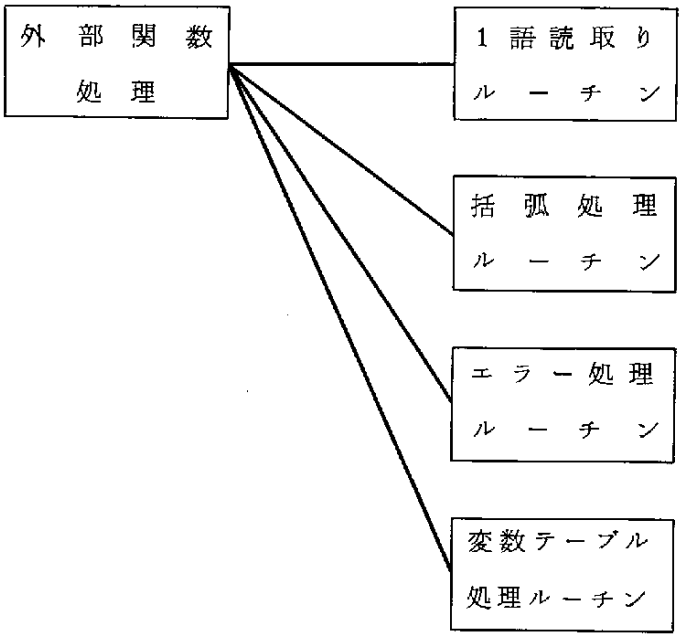
の時の変換した後の形式は次の様になる

IEXAM＝ISUM＋ISNMBR（０，▼ＳＥＴ１▼）＊１０

この時外部関数はISNMBR"("という事によって認知され，（ＳＥＴ１）を読込む事によって，（０，▼ＳＥＴ１▼）を作成しなければならず，その後に続く式＊１０をその後につけ加えなければならない。すなわち，（ＳＥＴ１）をそのまま変数として外部関数に手渡したのでは，引数のエラーとなるばかりでなく，文字ストリングとして効力を発揮しない。このため，UNGLプリ・コンパイラでは，外部関数のチェックだけでなく，変換も行ない，さらに後に続く文字に関しても誤りのないように処理する事が必要となる。

次にUNGLの外部関数処理モジュールの構造を示す。

〔図２－３０〕　外部関数処理モジュール構造



これらの外部関数は，実数値，整数値，あるいは論理値をとるものがあるが，それらの型について引用に注意しなければならない。

以下に外部関数の意味について述べるが，ａ～ｆ迄は論理値，ｇ，ｈは整数値，ｉは実数値として関数値を得る。なお，アーギュメントに示されるＩＴＥＭＥｎはアイテム・エクスプレッションを示し，ＳＥＴＥｎはセット・エクスプレッションを示す。

ａ．ＩＳＴＲＰＬ（ＩＴＥＭＥ１，ＩＴＥＭＥ２，ＩＴＥＭＥ３）

連想記憶にトリプルが存在するか否かを調べる。

実引数は，三つ組のアイテム・エクスプレッションであり，存在する時．ＴＲＵＥ．

ｂ．ＩＳＥＱＳＴ（ＳＥＴＥ１，ＳＥＴＥ２）

セット間のメンバーの対応を調べる。

SETE１とSETE２のメンバーが等しければ．TRUE．

d．ISMBER（ITEME１，SETE１）

あるセットにアイテムが含まれるか否かを調べる。

ITEME１がSETE２のメンバーであれば．TRUE．

e．ISITEM（ITEME１）

アイテムの存在を調べる。

ITEME１が存在すれば．TRUE．

f．LDATA（ITEME１，セル番号）

ITEME１で指定されたアイテムのセル番号目のデータを，論理数として得る。

g．ISNMBR（SETE１）

セットに含まれるメンバ数を関数値として得る。

h．IDATA（ITEME１，セル番号）

ITEME１で指定されたアイテムのセル番号目のデータを整数として得る。

i．RDATA（ITEME１，セル番号）

ITEME１で指定されたアイテムのセル番号目のデータを実数として得る。

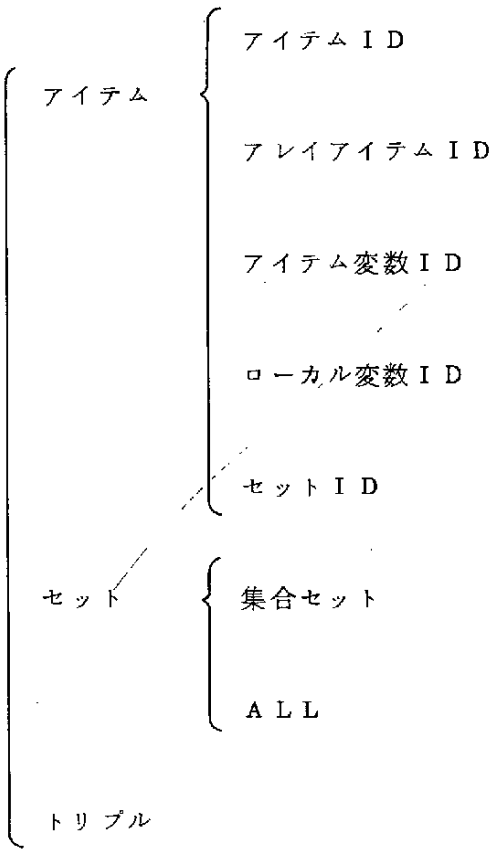
２．８　言語仕様とオブジェクトのまとめ

UNGLステートメントの言語仕様の主なものに関しては，「２．３」で述べた。ここで先ず図形データ処理，汎用データ・ストラクチャ操作で扱かわれるデータに関して〔図２－３１〕，〔図２－３２〕にまとめ，さらにそれらのデータ表現をＢＮＦで表わす。

〔図２－３１〕　図形データ表現

- グラフィック・データ
  - イメージ
    - ２次元イメージ
    - ３次元イメージ
  - イメージ・データ
    - ２次元イメージ・データ
    - ３次元イメージ・データ
  - 変換マトリックス
    - ２次元イメージ用座標変換マトリックス
    - ３次元イメージ用　〃　〃
    - 透視変換マトリックス
    - 平行投影マトリックス
  - 論理化因子
- グラフィック・オペレータ　{　⇐，＋，｜，.TRS2.，.TRS3.，.PERS.，.ORTH.}

〔図２－３２〕　汎用データ・ストラクチャ表現



a．図形データ表現

＜イメージγエクスプレッション＞：：＝＜イメージγターム＞｜＜＜イメージγターム＞｜＜論理化因子＞＞

＜イメージγターム＞：：＝＜イメージγファクタ＞｜（＜イメージγエクスプレッション＞＋＜イメージγエクスプレッション＞）

＜イメージ２ファクタ＞：：＝＜イメージ２プライマリ＞｜＜イメージ２エクスプレッション＞．ＴＲＳ２．＜座標変換

2マトリックス>|<イメージ3エクスプレッション>.PERS.<透視変換マトリックス>|<イメージ3エクスプレッション>.ORTH.<平行投影マトリックス>

<イメージ3ファクタ>::=<イメージ3プライマリ>|<イメージ3エクスプレッション>.TRS3.<座標変換3マトリックス>

<イメージrプライマリ>::=<イメージrID>|<イメージrデータ>

<イメージrIDリスト>::=<イメージrID>|<イメージrID>,<イメージrIDリスト>

<イメージrID>::=<英字名>

<イメージrデータ>::=<イメージrデータリスト>

<イメージrデータリスト>::=<rグラフィック・ファンクション>|<イメージrデータリスト>,<rグラフィック・ファンクション>

<2グラフィック・ファンクション>::=<LINE2ファンクション>|<TEXTファンクション>|<NUMBERファンクション>

<3グラフィック・ファンクション>::=<LINE3ファンクション>

<ムーヴィング情報>::=<(ムーヴィング・ブロック)>

<ウインドウイング・オプション>::=<イメージ領域>,<フレーム領域>|<イメージ領域>

<イメージ領域>::=<始点のx座標>,<始点のy座標>,<△x>,<△y>

<フレーム領域>::=<始点のX座標>,<始点のY座標>,<△X>,<△Y>

<論理化因子>::=<<論理名>,<制御情報>>|<<論理名>>

＜論理名＞：：＝＜アイテムＩＤ＞｜＜アイテム変数ＩＤ＞｜＜整数＞｜
＜実数＞｜＜ホラリスコンスタント＞

＜制御情報＞：：＝－３｜－２｜－１｜０｜１｜２｜３｜＜整変数＞

＜フレーム番号＞：：＝０｜１｜２｜……｜９９

b．汎用データ・ストラクチャ表現

＜アイテム・エクスプレッション＞：：＝＜アイテムＩＤ＞｜＜アイテム変数ＩＤ＞｜＜ローカル変数ＩＤ＞
｜＜アレイアイテムＩＤ＞

＜アイテム・エクスプレッション・リスト＞：：＝＜アイテム・エクスプレッション＞｜＜アイテム・エクスプレッション＞｜
＜アイテム・エクスプレッション・リスト＞

＜アイテム・プライマリリスト＞：：＝＜アイテム・プライマリ＞｜＜アイテム・プライマリ＞｜＜アイテム・プライマリリスト＞

＜アイテム・プライマリ＞：：＝＜アイテム・プライマリＩＤ＞／＜データ長＞／｜＜アイテム・プライマリＩＤ＞

＜アイテム・プライマリＩＤ＞：：＝＜アイテムＩＤ＞｜＜アレイ・アイテムＩＤ＞

＜データ長＞：：＝＜整定数＞｜＜整変数＞

＜アイテム変数ＩＤリスト＞：：＝＜アイテム変数ＩＤ＞｜＜アイテム変数ＩＤ＞，＜アイテム変数ＩＤリスト＞

＜ローカル変数ＩＤリスト＞：：＝＜ローカル変数ＩＤ＞｜＜ローカル変数ＩＤ＞，＜ローカル変数ＩＤリスト＞

＜セット・エクスプレション＞：：＝＜セット・ターム＞｜＜セット・エクスプレション＞．＊．＜セット・ターム＞

＜セット・ターム＞：：＝＜セット・ファクタ＞｜＜セット・ターム＞．＋
．＜セット・ファクタ＞

＜セット・ファクタ＞：：＝＜セット・プライマリ＞｜＜セット・ファクタ＞
．－．＜セット・プライマリ＞

＜セット・プライマリ＞：：＝＜セットＩＤ＞｜ＡＬＬ

＜セットＩＤリスト＞：：＝＜セットＩＤ＞｜＜セットＩＤ＞，＜セットＩ
Ｄリスト＞

＜トリプル＞：：＝（＜アイテム・エクスプレション＞，＜アイテム・エク
スプレション＞，＜アイテム・エクスプレション＞）

＜セル番号＞：：＝＜整変数＞｜＜整定数＞

c．割込み処理用データ

＜割込み要因リスト＞：：＝＜割込み要因＞｜＜割込み要因＞，＜割込み要
因リスト＞

＜割込み要因＞：：＝ＬＰＥＮ｜ＴＲＣＭ｜ＦＫＥＹ｜ＭＶＥＤ｜
ＤＩＡＬ｜ＭＥＳＧ

＜ステート番号＞：：＝１｜２｜３｜……９９

＜処理プログラム名＞：：＝＜英字名＞

２．８．１　言語仕様

UNGL言語仕様の全体をＢＮＦで示す。

a．図形データ処理用ステートメント

＜IMAGEステートメント＞：：＝IMAGE＊２＜イメージＩＤリスト＞
｜IMAGE＊３＜イメージＩＤリスト＞

＜イメージ代入ステートメント＞：：＝＜イメージｒＩＤ＞＜＝＜イメージ
ｒエクスプレション＞

＜DRAWステートメント＞：：＝DRAW＜フレーム番号＞，＜イメージ２

ＩＤリスト＞（＜ウインドウイング・オプション＞）｜DRAW＜フレーム番号＞，＜イメージ２ＩＤリスト＞

＜ANIMATEステートメント＞：：＝ANIMATE＜イメージ２ＩＤリスト＞（＜ムーヴィング情報＞）｜ANIMATE＜イメージ２ＩＤリスト＞（＜ムーヴィング情報＞）（＜ウインドウイング・オプション＞）

＜DELETEステートメント＞：：＝DELETE＜フレーム番号＞，＜論理化因子＞

＜CONTROLステートメント＞：：＝CONTROL＜フレーム番号＞，＜論理化因子＞

＜RESETステートメント＞：：＝RESET＜フレーム番号＞

＜DISPLAYステートメント＞：：＝DISPLAY＜フレーム番号＞

b．汎用データ・ストラクチャ操作用ステートメント

＜ITEMステートメント＞：：＝ITEM＜アイテム・プライマリリスト＞

＜ITEMVARステートメント＞：：＝ITEMVAR＜アイテム変数ＩＤリスト＞

＜SETステートメント＞：：＝SET＜セットＩＤリスト＞

＜LOCALステートメント＞：：＝LOCAL＜ローカル変数ＩＤリスト＞

＜アイテム代入文＞：：＝＜アイテム変数ＩＤ＞＜－＜アイテムプライマリＩＤ＞

＜ALLOCステートメント＞：：＝ALLOC＜アイテム変数ＩＤ＞，＜配列名＞，＜データ長＞，｜ALLOC＜アイテム変数ＩＤ＞，＜配列名＞

＜FREEステートメント＞：：＝FREE＜アイテム・エクスプレション・リスト＞

＜MAKEステートメント＞：：＝MAKE＜トリプル＞

＜ERASEステートメント＞：：＝ERASE＜トリプル＞

＜PUTステートメント＞：：＝PUT（＜アイテム・エクスプレション・リスト＞）ＩＮ＜セットＩＤ＞｜PUT（＜セットエクスプレション＞）ＩＮ＜セットＩＤ＞

＜REMOVEステートメント＞：：＝REMOVE（＜アイテム・エクスプレション・リスト＞）FROM＜セットＩＤ＞｜REMOVE（＜セット・エクスプレション＞）FROM＜セットＩＤ＞

＜IDATAステートメント＞：：＝IDATA（＜アイテム・エクスプレション＞，＜セル番号＞）

＜RDATAステートメント＞：：＝RDATA（＜アイテム・エクスプレション＞，＜セル番号＞）

＜LDATAステートメント＞：：＝LDATA（＜アイテム・エクスプレション＞，＜セル番号＞）

＜FOREACHステートメント＞：：＝FOREACH（＜ローカル変数ＩＤ＞）ＩＮ＜セット・エクスプレション＞ＤＯ＜ステートメント番号＞｜FOREACH＜条件文＞ＤＯ＜ステートメント番号＞

c．割込み処理用ステートメント

＜DEFINESステートメント＞：：＝DEFINES

＜ENDSステートメント＞：：＝ENDS

＜STATEステートメント＞：：＝STATE＜ステート番号＞

＜ＯＮ定義文＞：：＝ＯＮ＜割込み要因＞ＤＯ＜処理プログラム名＞，
＜ステート番号＞｜ＯＮ＜割込み要因＞ＤＯ＜処
理プログラム名＞

＜ＯＮ実行文＞：：＝ＯＮ＜割込み要因＞ＤＯ＜処理プログラム名＞

＜ＴＲＡＮＳＦＥＲステートメント＞：：＝ＴＲＡＮＳＦＥＲ＜ステート番号＞

＜ＡＷＡＩＴステートメント＞：：＝ＡＷＡＩＴ

＜ＡＴＡＫＥステートメント＞：：＝ＡＴＡＫＥ＜変数名＞，＜割込み要因リス
ト＞

＜ＡＥＸＩＴステートメント＞：：＝ＡＥＸＩＴ

＜ＯＮ変更文＞：：＝ＯＮ＜割込み要因＞ＤＯ＜処理プログラム名＞，，
＜ステート番号＞｜ＯＮ＜割込み要因＞ＤＯ＜処理プ
ログラム名＞，＜ステート番号＞，＜ステート番号＞

＜ＬＯＣＫステートメント＞：：＝ＬＯＣＫ＜割込み要因＞，＜ステート番号＞
｜ＬＯＣＫ＜割込み要因＞

＜ＵＮＬＯＣＫステートメント＞：：＝ＵＮＬＯＣＫ＜割込み要因＞，＜ステー
ト番号＞｜ＵＮＬＯＣＫ＜割込み要因＞

＜ＳＰＥＣＩＦＹステートメント＞：：＝ＳＰＥＣＩＦＹ＜配列名＞

d．外部関数

＜ＩＳＴＲＰＬファンクション＞：：＝ＩＳＴＲＰＬ（＜トリプル＞）

＜ＩＳＳＢＳＴファンクション＞：：＝ＩＳＳＢＳＴ（＜セット・エクスプレシ
ョン＞，＜セット・エクスプレション＞）

＜ＩＳＥＱＳＴファンクション＞：：＝ＩＳＥＱＳＴ（＜セット・エクスプレシ
ョン＞，＜セット・エクスプレション＞）

＜ＩＳＭＢＥＲファンクション＞：：＝ＩＳＭＢＥＲ（＜アイテム・エクスプレ
ッション＞，＜セット・エクスプレッ
ション＞）

＜ＩＳＩＴＥＭファンクション＞：：＝ＩＳＩＴＥＭ（＜アイテム・エクスプレ
ッション＞）

＜ＬＤＡＴＡファンクション＞：：＝ＬＤＡＴＡ（＜アイテム・エクスプレッ
ション＞，＜セル番号＞）

＜ISNMBRファンクション＞：：＝ISNMBR（＜セット・エクスプレッション＞）

＜IDATAファンクション＞：：＝IDATA（＜アイテム・エクスプレッション＞，＜セル番号＞）

＜RDATAファンクション＞：：＝RDATA（＜アイテム・エクスプレッション＞，＜セル番号＞）

2．8．2　オブジェクト

前記のステートメントによって作成されるオブジェクトは，FORTRANソース・ステートメントであり，FOREACH文と外部関数を除いてCALL文となる。次にその一覧表を示す。

a．図形データ処理用

UNGL文		FORTRAN・CALL文	備　　考
IMAGE文		S¥IMGD(次元, ▼イメージID▼〔, ▼イメージID▼..〕)	i：イメージ再定義指標 0＝平常定義 1＝再定義
イメージ代入文	＋	S¥IMAD(po, {▼イメージID▼ / インターナルID}, i, {▼イメージID▼ / インターナルID}, i)	
	−	S¥ARGL(po, {1, ▼アイテムID▼, n / 0, 論理名}, 制御情報) S¥GFAC(p1, {▼イメージID▼ / インターナルID}, i, po)	j：変換マトリックス指標 1＝座標変換 2マトリックス 2＝座標変換 3マトリックス 3＝透視変換 マトリックス 4＝平行投影 マトリックス
	.TRS2. .TRS3. .PERS. .ORTH.	S¥ARGL(po, j, 変換マトリックス) S¥MATX(p1, {▼イメージID▼ / インターナルID}, i, po)	
	⇐	S¥ASGN(▼イメージID▼, {▼イメージID▼ / インターナルID})	
	イメージ・データ	S¥ARGL(po, pn, k, イメージ・データ) S¥IMDT(po, p1〔, p2・・・pn〕)	k：グラフィック・ファンクション 1＝LINE2 2＝LINE2　配列 3＝LINE3 4＝LINE3　配列 5＝TEXT 6＝NUMBER
DRAW文		S¥WIND(ウインド情報) S¥DRAW({{フレーム番号}, ▼イメージID▼〔, ▼イメージID▼・・〕}) S¥WIND(ウインド情報)	

ANIMATE文	S¥ANIM(▼イメージID▼〔, ▼イメージID▼・・・〕) S¥MVGI(ムービング情報) S¥ADRW	pn：中間結果用変数 インターナルID： イメージの中間結果 n：アレイ・アイテム用 指標
DELETE文	S¥DELT({0, フレーム番号}, {1, ▼アイテムID▼, n / 0, 論理名}, 制御情報)	
CONTROL文	S¥CNTL({0, フレーム番号}, {1, ▼アイテムID▼, n / 0, 論理名}, 制御情報)	
RESET文	S¥REST({0, フレーム番号})	
DISPLAY文	S¥DSPL(フレーム番号)	

b．汎用データ・ストラクチャ操作用

ITEM文	H¥ITEM(▼アイテムID▼, データ長, 配列添字)
ITEMVAR文	
SET文	H¥SETD(▼セットID▼[, ▼セットI ▼・・・])
LOCAL文	
アイテム代入文	H¥ALLC(アイテム変数ID, ▼アイテムID▼, 配列添字)
ALLOC文	H¥ALLC(アイテム変数ID, 配列名, データ長)
FREE文	H¥FREE({▼アイテムID▼ / アイテム変数ID}, n, [, {▼アイテムID▼ / アイテム変数ID}, n・・・])
MAKE文	H¥MAKE({▼アイテムID▼ / アイテム変数ID}, n, {▼アイテムID▼ / アイテム変数ID}, n, {▼アイテムID▼ / アイテム変数ID}, n)
ERASE文	H¥ERSE(triple)
PUT文	H¥PUTM(▼セットID▼, j, {{▼アイテムID▼ / アイテム変数ID}, n[, {▼アイテムID / アイテム変数ID}, n・・] / ▼セットエクスプレッション▼})
REMOVE文	H¥REMV(▼セットID▼,[j, {{▼アイテムID▼ / アイテム変数ID}, n[, {▼アイテムID / アイテム変数ID}, n・・]D) / ▼セット・エクスプレッション▼}

n：アレイアイテムの配列添字
0＝アイテム変数
1＝アイテム
正＝アレイ・アイテム
j：セット用指標
0又は負＝集合演算子を含まないセット
正＝集合演算子をj個含むセット

セット・エクスプレッション
j≦0の時
▼セットID▼
j＞0の時
▼セットID＊セットID▼
の様に集合演算子使用
＊tripleはH¥MAKEの(　)の中と同様な形式

IDATA文 RADTA文 LDATA文	H¥DATA({▼アイテムID▼ アイテム変数ID}, n, セル番号, 右辺エクスプレッション)	*tripleはH¥MAKEの( )の中と同様な形式 is, ie, issシステム用作業名 m { 0＝セット表現 1＝triple表現 *tripleはH¥MAKEの( )の中と同様な形式
FOREACH文	H¥ESTM(is, ie, iss, m, {j, セット・エクス・プレッション triple}) DO ステートメント番号 ¥i＝is, ie〔, 2〕	
ISTRPL関数	ISTRPL(triple)	
ISSBST関数	ISSBST(j, ▼セット・エクスプレッション▼, j, ▼セット・エクスプレッション)	
ISEQST関数	ISEQST(j, ▼セット・エクスプレッション▼ j, ▼セット・エクスプレッション)	
ISMBER関数	ISMBER({▼アイテムID▼ アイテム変数ID}, n, j, ▼セット・エクスプレッション▼)	
ISITEM関数	ISITEM({▼アイテムID▼ アイテム変数ID}, n)	
LDATA関数	LDATA({▼アイテムID▼ アイテウ変数ID}, n, セル番号)	
ISNMBR関数	ISNMBR(j, ▼セット・エクスプレッション▼)	
IDATA関数	IDATA({▼アイテムID▼ アイテム変数ID}, n, セル番号)	
RDATA関数	RDATA({▼アイテムID▼ アイテム変数ID}, n, セル番号)	

c．割込み処理用

DEFINES文	K¥INIT	
ENDS文	K¥ENDS	
STATE文	K¥STAT（ステート番号）	
ON定義文	K¥ONST〔0，割込み要因，処理ルーチン名〔，ステート番号〕）	
ON実行文	K¥ONST（1，割込み要因，処理ルーチン名，0）	
TRANSFER文	K¥TRNS（ステート番号）	
AWAIT文	K¥WAIT	
ATAKE文	K¥TAKE（変数名，割込み要因〔，割込み要因・・・〕）	ATAKE文の 割込み要因は6個迄記述可
AEXIT文	K¥EXIT	
ON変更文	K¥ONST（1，割込み要因，処理ルーチン名，ステート番号，ステート番号）	
LOCK文	K¥LOCK（割込み要因〔，ステート番号〕）	
UNLOCK文	K¥UNLK（割込み要因〔，ステート番号〕）	
SPECIFY文	K¥SPEC（配列名）	

3. 図形データ処理

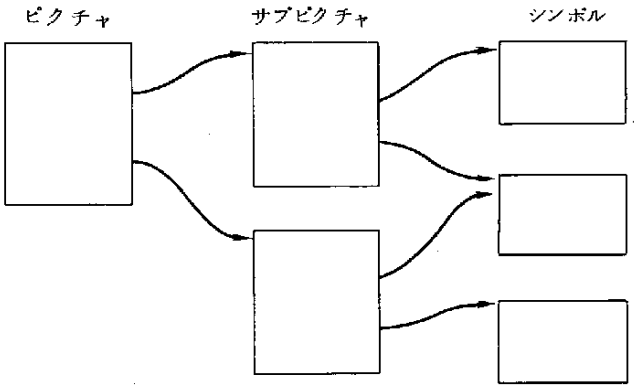
3. 図形データ処理

3.1 図形データ処理の概念

図形をコンピュータで扱う場合，ディスプレイ画面に表示されるデータをどのように記憶するかが問題となる。

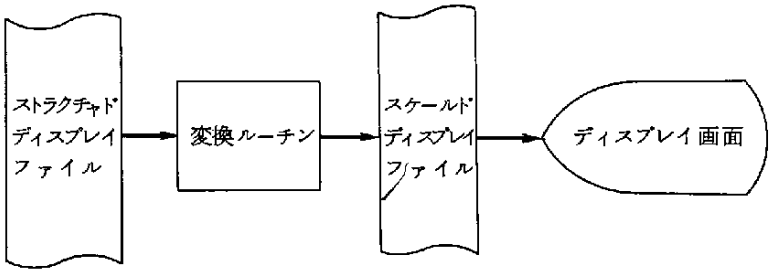
リフレッシュ・タイプ（refresh type）のディスプレイ装置では，ディスプレイ・プロセッサがディスプレイ・ファイルと呼ばれるディスプレイ・コマンド列を繰り返し実行することにより，画面がリフレッシュされ図形が表示される。ディスプレイ・コマンドはビームのポジショニング（positioning）あるいはベクトル表示などの座標データや表示のための制御情報を含む。またブランチ命令，ディスプレイ・サブルーチン・コール，リンク命令なども含んでいる。しかし画面の一部に修正を加えたり，一部を除去したりすることを可能にするには，何らかの形でディスプレイ・ファイルにアクセスができなくてはならない。またディスプレイ・ファイルの中にサブピクチャを定義したりする必要性もあるだろう。この目的からディスプレイ・ファイルは適当に構造化され扱われる必要がある。これは一般にストラクチャド・ディスプレイ・ファイル（structured display file）と呼ばれている。ストラクチャド・ディスプレイ・ファイルは単なる線型リストではなく一般に〔図３－１〕のようなトリー構造をとる。

〔図3－1〕



ディスプレィ・プロセッサはこの構造をたどり，ブランチ命令あるいはサブルーチン・コールを解読しながらリフレッシュを行なう。ディスプレィ・ファイルは構造化されているために，一部分を変更する際も他の部分に影響を与えることもなく図形の修正が容易にできる。また繰り返し生じるシンボルはサブルーチンとして表現することができるため，ディスプレィ・ファイルのデータ量を少なくすることができる。さらにライトペンのようなポインティング・ディバイス（pointing device）から構造の一部分を指摘することも可能である。これらの特性はインタラクティブ・グラフィック・システムにとっては不可欠なものである。しかしこのストラクチャド・ディスプレィ・ファイルにも欠点がある。一つはマシン・インディペンデントな扱いが困難であるということである。例えば汎用グラフィック言語でこれをどのように扱うかというのは難しい問題である。次にリフレッシュができるだけ速く行えるようなディスプレィ・ファイルの設計をしなければならないということである。さらにまたこれらの処理はディスプレィ装置にディペンドするためその装置がもつ機能により制限がでてくる。例えば，ディスプレィ・プロセッサにスケーリング機能がなければ，図形は常に決ったスケールで表示されてしまう。そのためにスケーリング機能は〔図３－２〕に示すようにソフトウェアに組み入れなければならない。

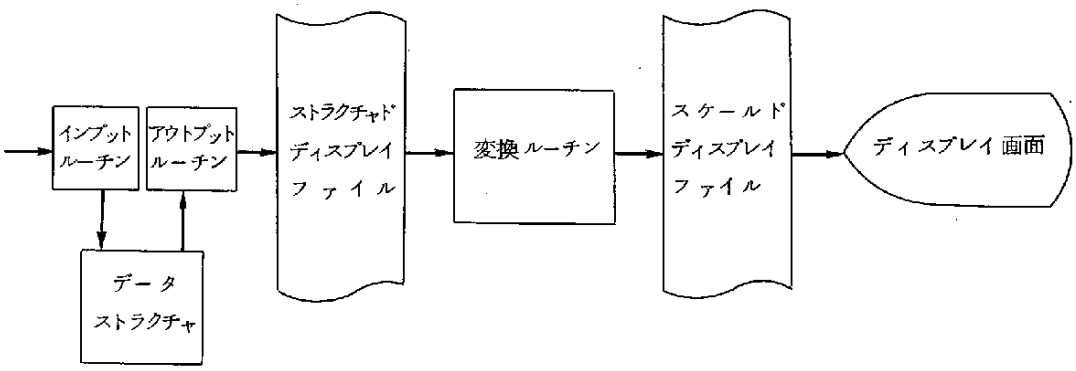
〔図3－２〕



ここではストラクチャド・ディスプレィ・ファイルはマスタ・コピーとして存在し，スケールド・ディスプレィ・ファイル（scaled display file）が画面のリフレッシュをつかさどっている。

さらに汎用のグラフィック・システムでは，ユーザがデータ・ストラクチャを形成できるようなファシリティが必要とされるだろう。このストラクチャはプログラムのインプット・ルーチンによりデータが与えられ，さらにアウトプット・ルーチンによりアクセスされディスプレィ・ファイルが形成される。このように汎用グラフィック・システムでは一般に〔図３－３〕のような構造がとられる。

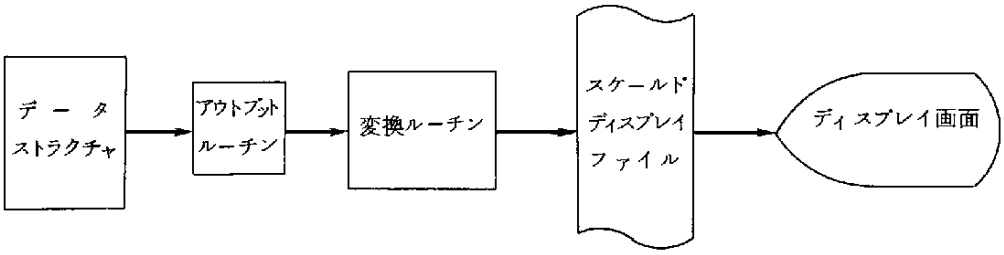
〔図3－3〕



ここでインタラクションを速くすることを考えてみると，まずスケールド・ディスプレィ・ファイルはディスプレィ・プロセッサの機能を拡大することにより取り除くことができるが，そのためには高価な端末とディバイス・インディペンデント（device independent）な言語が必要となる。またストラクチャド・ディスプレィ・ファイルのピクチャの回転やスケーリングの情報をディスプレィ・ファイルに含めることもできるが，それなりのハードウェアを必要とするだろう。これらのことを考慮に入れると一般には〔図３－４〕に示すようなストラクチャド・ディスプレィ・ファイルを省いたグラフィッ

ク・システムが良いとされている。

〔図３－４〕



このようにストラクチャド・ディスプレイ・ファイルを取り除いたために，図形データはアウトプット・ルーチンにより，スケーリングなどの変換ルーチンを経て直接スケールド・ディスプレイ・ファイルに出力される。インタラクティブ・グラフィックスにおいてはこのシステムが有効であるとされているが，しかし２つの欠点がある。まず図形に修正を加えた時には，このスケールド・ディスプレイ・ファイル全体を修正しなければならない。もう一つはポインティング・ディバイスからのインプットすなわち▼ヒット▼と呼ばれている処理の手段を考えなければならない。

UNGLではこれらのことからインタラクションの速度を重視し，〔図３－４〕に示すようなものに近い型式をとっている。

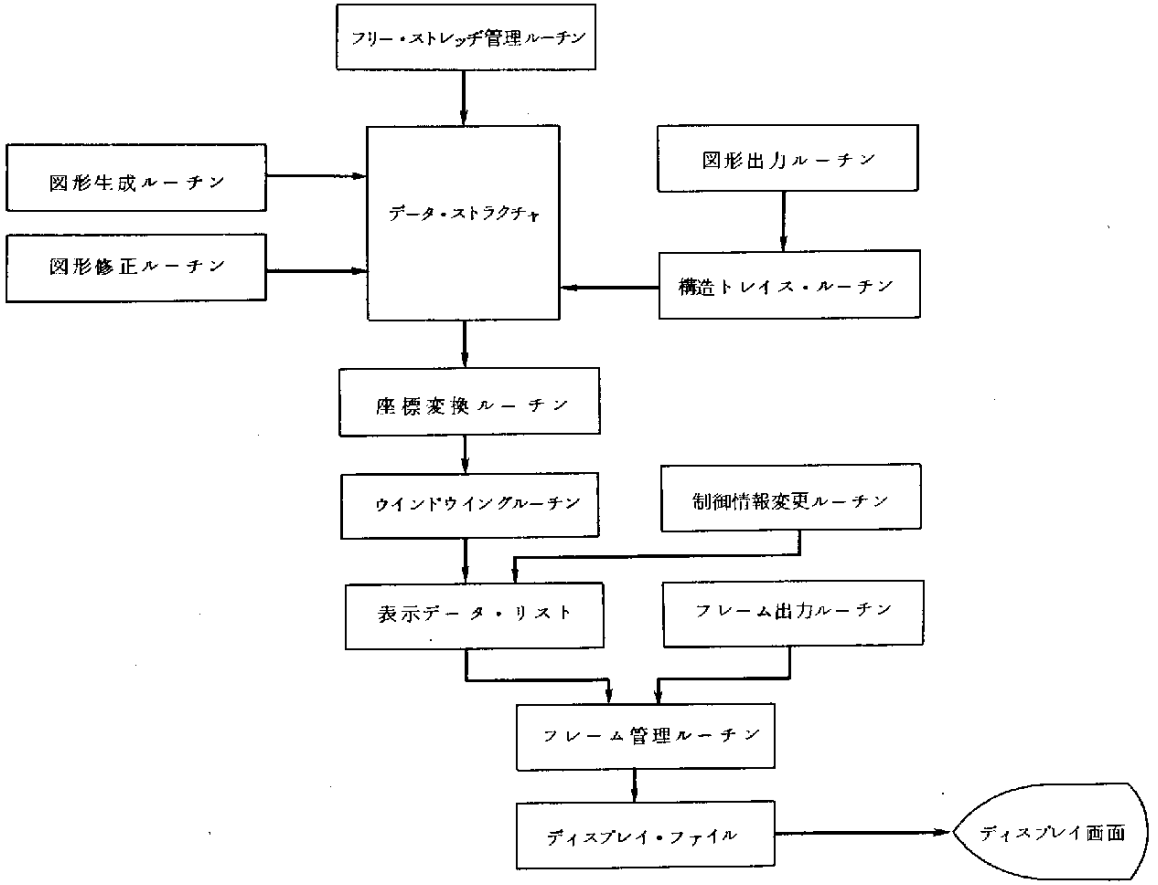
3.2 UNGLにおける図形データ処理

ＵＮＧＬでは２次元，３次元の図形を扱うことができる。図形を定義し，さらにそれらの図形に回転，拡大，平行移動，投象あるいは図形の併合などの操作が容易にできるような図形生成，修正ステートメントが用意されている。生成された図形はトリー構造のデータ・ストラクチャの形で存在する。そのために実行時に図形の修正が行われると，このデータ・ストラクチャが修正されることになる。図形を出力する場合は，このデータ・ストラクチャをトレイス（trace）することにより，対象となる図形のデータを図形変換の操作を行いながらとりだす。この時データ・ストラクチャで表わされている図形とディスプレィ画面に表示したい図形とは座標系が異なるためにその変換も必要となる（ウィンドウイング）。トレイス後にできる表示データ・リストは基本的な図形要素（線あるいは文字）を識別したい図形単位に集めたものである。この各図形要素はディスプレィ・コマンド列に翻訳され，ディスプレィ・ファイルが生成される。そしてこのディスプレィ・コマンド列が実行されディスプレィ画面に図形が表示される。

また表示される図形は画面単位の表示データ・リストの形で保存しておくことができる。さらにその保存されている図形に対して修正を加えることもできるし，再びその画面を表示することもできる。

次にＵＮＧＬにおける図形処理の流れを〔図３－５〕に示す。

〔図３－５〕


フリー・ストレッチ管理ルーチン
図形生成ルーチン
図形修正ルーチン
データ・ストラクチャ
図形出力ルーチン
構造トレイス・ルーチン
座標変換ルーチン
ウインドウイングルーチン
制御情報変更ルーチン
表示データ・リスト
フレーム出力ルーチン
フレーム管理ルーチン
ディスプレイ・ファイル
ディスプレイ画面

3.2.1 図形データ

図形をコンピュータで処理するにあたって，まず問題となるのは図形データの表現法である。例えば2次元内のある線分は両端点の座標値で表現することもできるし，直線方程式によって表現することもできる。型式は異なれ一般には2次元の図形は点と線の関連によって表現でき，3次元の図形は点，線，面の関連によって表現できる。

しかし実際に画面に表示する要素は線分であるということからUNGLでは図形表示の要素としては，2次元，3次元ともに線分のみをとりあげて，両端点の座標値による表現法をとっている。例えば〔図3－6〕のような立方体は，

① (0, 0, 0), (1, 0, 0), (1, 1, 0), (0, 1, 0), (0, 0, 0)
(0, 0, 1), (1, 0, 1), (1, 1, 1), (0, 1, 1), (0, 0,1),

② (0, 1, 0), (0, 1, 1)

③ (1, 1, 0), (1, 1, 1)

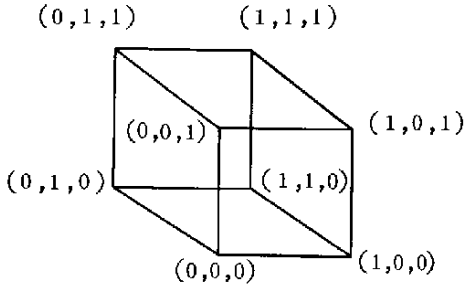
④ (1, 0, 0), (1, 0, 1)

①～④の順序に各座標を結ぶ直線で表わす。又これは次のようなUNGLステートメントで表わされる。

```
LINE3(X, Y, Z, 1, 10), LINE3(0, 1, 0, 0, 1, 1),
LINE3(1, 1, 0, 1, 1, 1), LINE3(1, 0, 0, 1, 0, 1)
```

ここでLINE3は3次元の線分を定義するグラフィック・ファンクションである。

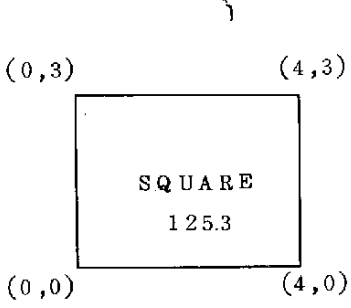
〔図3－6〕



このように図形の実体を定義する図形表示の要素をイメージ・データと呼ぶ。イメージ・データを表わすグラフィック・ファンクションには他に２次元の線分を表わすＬＩＮＥ2，文字を表わすＴＥＸＴ，数字を表わすＮＵＭＢＥＲがある。

例えば〔図３－７〕は次のように表現される。

〔図3－7〕




```
'LINE2(0,0,4,0,4,3,0,3,0,0),
 TEXT ('SQUARE', 1,1,6,1),
 NUMBER(125.3,1.5,0.5,F5.1,0)'
```

ここでＬＩＮＥ2（０，０，４，０，４，３，０，３，０，０）は（０，０），（4,0），（４，３），（０，３）を頂点とする四角形を定義し，ＴＥＸＴ（'ＳＱＵＡＲＥ'，１，１，６，１）は'ＳＱＵＡＲＥ'という６文字の文字ストリングを始点（１，1）から大文字で表わすことを定義し，さらにＮＵＭＢＥＲ（１２５.３，1.5，0.5，Ｆ5.1，０）は１２５.３という数字を始点（1.5，0.5）から小文字でＦ5.1のフォーマットに従って表わすことを定義している。

これらのイメージ・データによって表現された図形は，他の図形を表現するのに利用することもできる。例えば〔図３－８〕のような連続図形を描きたい時には，一つの三角形のイメージ・データを定義し，その三角形を平行移動させることによって表現できる。さらにこの連続図形もまた他の図形の表現手段として使用できる。イメージにはまた他のイメージを含むこともできる。

〔図3－8〕

3.2.2 図 形 変 換

イメージを表現する場合，前述のように他のイメージを回転させたり，拡大・縮少したり，又平行移動したりすることにより定義することが多い。又3次元のイメージはそのまま画面に表示することができないために，そのイメージを回転させてある視点から平面に投象するような方法も必要であろう。そしてこれらの動作は座標を同次座標（Homogeneous Coordinate）で表わし，マトリックスをかけあわせることにより表現できることは，よく知られている。

A. 2次元の座標変換

2次元の点（x，y）は同次座標（x，y，1）で表現され，3×3の変換マトリックスMをかけあわせることにより，他の点（x′，y′）に変換される。

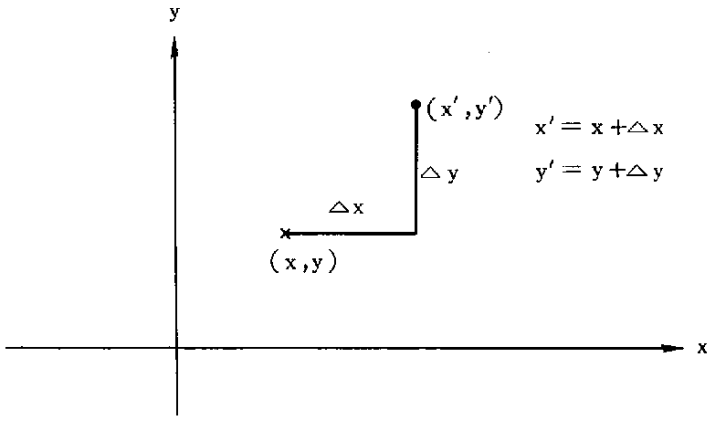
$$(x', y', 1) = (x, y, 1)M$$

(1) 平行移動

点（x，y）を（x+△x，y+△y）に変換したい場合，変換マトリックスは次のように表わすことができる。

$$(x', y', 1) = (x, y, 1) \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \triangle x & \triangle y & 1 \end{pmatrix}$$

〔図3－9〕

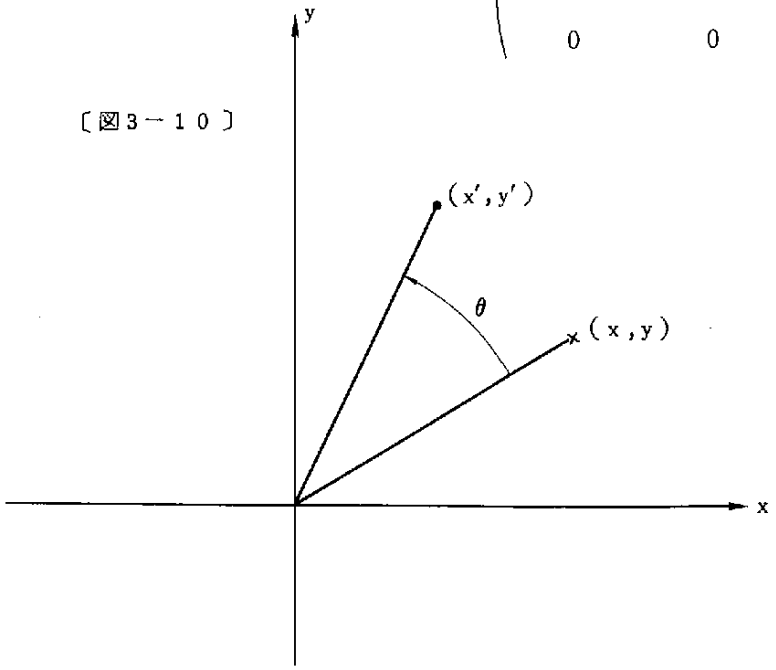


(2) 回　　転

点（ｘ，ｙ）を原点中心に角度θだけ回転させた点を（ｘ′，ｙ′）とすると変換マトリックスは次のように表わすことができる。

$$(x', y', 1) = (x, y, 1) \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

〔図3－10〕

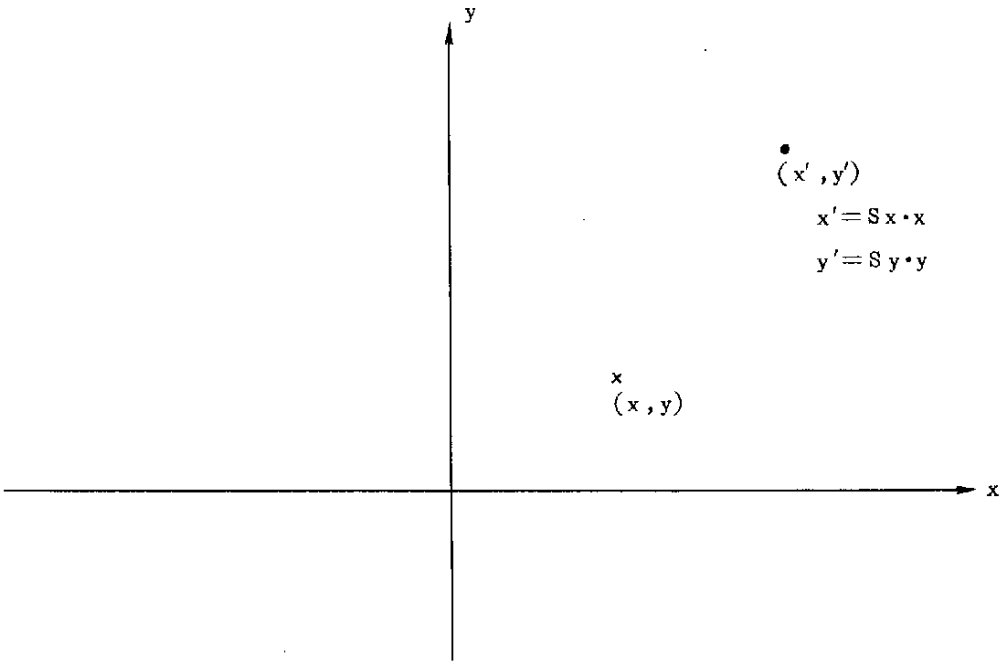


(3) 拡大・縮小

点（x，y）をx方向にSx ，y方向にSy ，拡大した場合の点（x′,y′)は次のように表わされる。

$$(x',y',1)=(x,y,1)\begin{pmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

〔図3－11〕



又，x，y両方向への拡大は次のようにも表現できる。

$$(x',y',1)=(x,y,1)\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1/S \end{pmatrix}$$

B． 3次元の座標変換

3次元の点（x，y，z）は同次座標（x，y，z，1）で表現され，4×4の変換マトリックスMをかけあわせることにより，他の点（x′，y′，z′）に変

換される。

$$(x', y', z', 1) = (x, y, z, 1)M$$

変換マトリックスMは２次元の座標変換の時と同様に平行移動，回転，拡大・縮小を表わす。

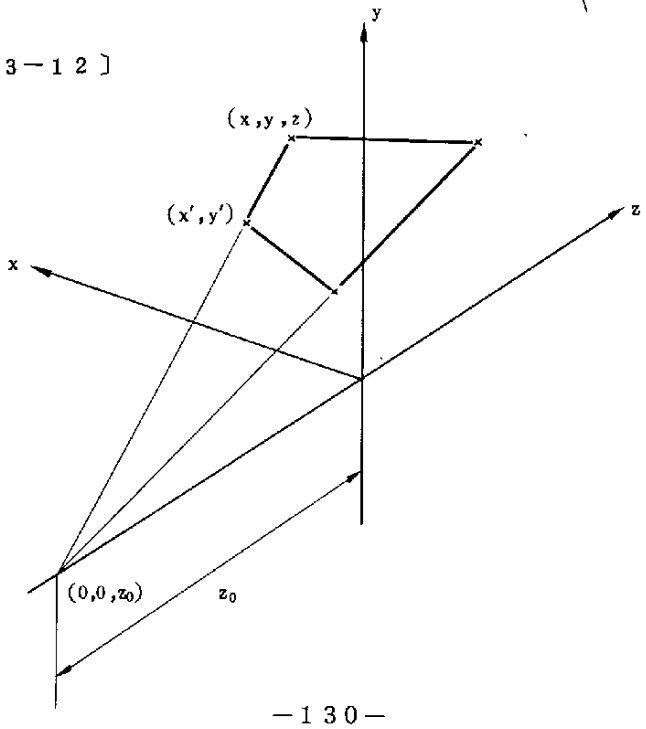
$$M = \left(\begin{array}{c|c} R & P \\ \hline T & S \end{array}\right)$$

ここでRは３×３の回転変換マトリックス，Tは平行移動を表わす行ベクトル，さらにSは拡大・縮小を表わしている。

さらにPは透視変換を表わす列ベクトルである。３次元の点（ｘ，ｙ，ｚ）をｚ軸上の点（０，０，z_0）を視点としてｘ－ｙ平面に透視した点（ｘ′，ｙ′）とすると変換マトリックスは次のように表わされる。

$$(x', y', z', 1) = (x, y, z, 1) \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1/z_0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

〔図３－１２〕



又ｘ－ｙ平面への平行投影図は次の変換マトリックスをかけることによって求められる。

$$(x', y', z', 1) = (x, y, z, 1) \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

同様にｙ－ｚ平面，ｚ－ｘ平面への透視図，平行投影図を得る変換マトリックスも考えられる。

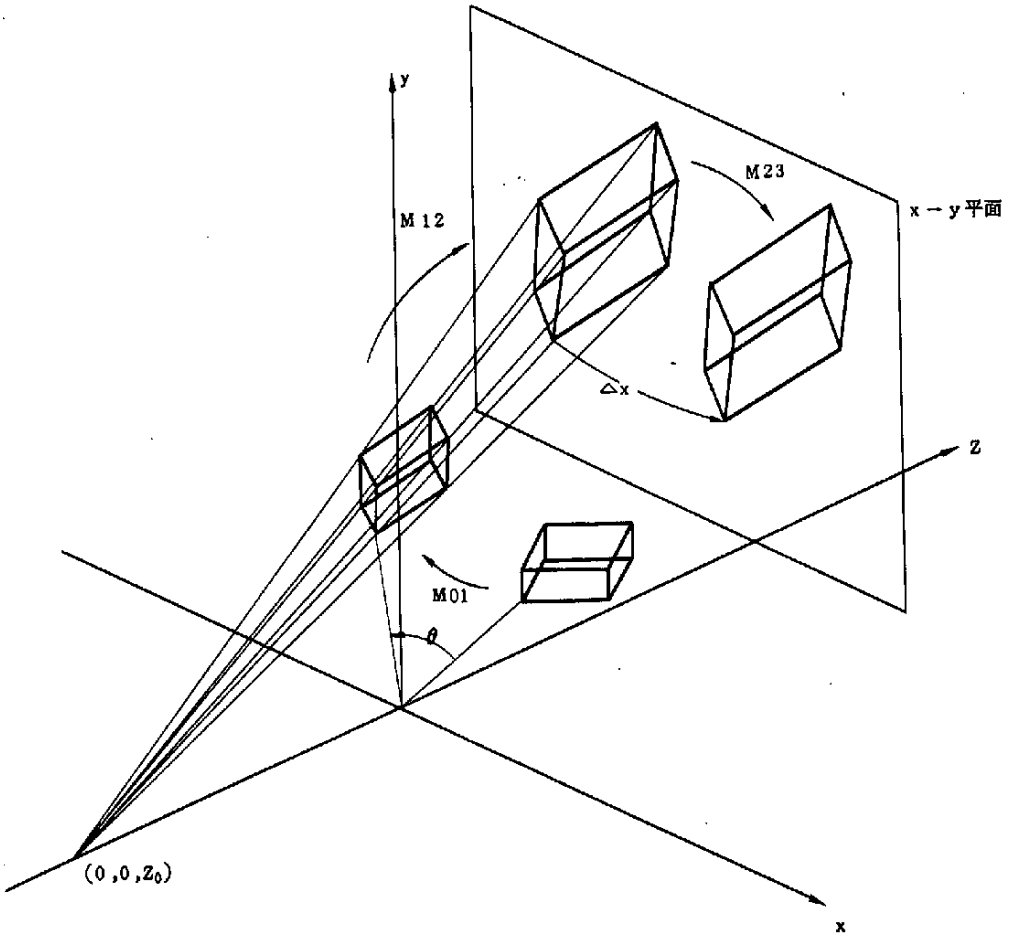
これらの動作は単独に行われることよりも回転した後で投象するなどというように連続的に行われることが多い。その際各変換マトリックスを順次かけあわせることにより連続的に変換操作を行うことができる。変換マトリックス M_{01}，M_{12}，M_{23}，M_{34}，M_{45} に対応する座標変換を順次行った場合には次式がなりたつ。

$$(x', y', z', 1) = (x, y, z, 1)\, M_{05}$$

$$M_{05} = M_{01} \cdot M_{12} \cdot M_{23} \cdot M_{34} \cdot M_{45}$$

例えば〔図３－１３〕に示すように３次元図形の直方体をｘ－ｙ平面に関して角度 θ の回転をする。その図形を（０，０，z_0）からｘ－ｙ平面に投象し，さらにｘ－ｙ平面上でｘ方向に $\triangle x$ の平行移動を施すとする。

〔図3－13〕



各変換に対する変換マトリックスは次のように表わされる。

$$M_{e1}=\begin{pmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad M_{12}=\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1/z_0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$M_{23} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \triangle x & 0 & 1 \end{pmatrix}$$

従って変換後の各点の座標は次のようにして求められる。

$$(x', y', z', 1) = (x, y, z, 1) M_{01} \cdot M_{12} \cdot M_{23}$$

$$= (x, y, z, 1) \begin{pmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1/z_0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \triangle x & 0 & 0 & 1 \end{pmatrix} = (x, y, z, 1) \begin{pmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ -\frac{\triangle x}{z_0} & 0 & 0 & -\frac{1}{z_0} \\ \triangle x & 0 & 0 & 1 \end{pmatrix}$$

$$x' = (\cos\theta \ x - \sin\theta \cdot y - \triangle x \cdot z / z_0 + \triangle x) \cdot z / (z_0 - z)$$

$$y' = (\sin\theta \cdot x + \cos\theta \cdot y) \cdot z / (z_0 - z)$$

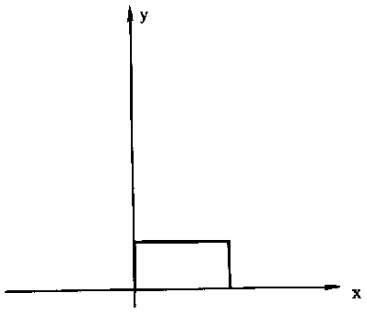
C．変換マトリックス

UNGLでは，ユーザが，変換の数学的な機構を意識することなく簡単に図形の座標変換を行うことができるように変換マトリックスを設けた。変換マトリックスはマトリックスそのものではなく，単に変換量を与えるだけであり，グラフィック・オペレータ　.TRS.，.PERS.，.ORTH. によってイメージに作用させることができる。

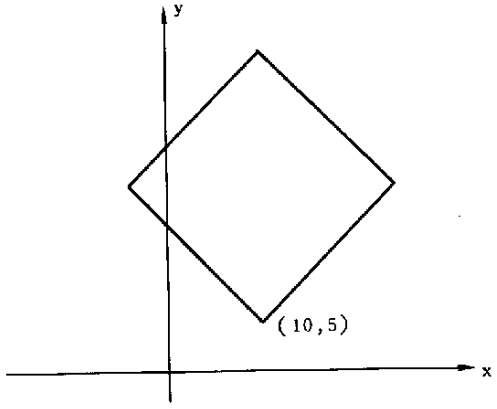
(1)　2次元座標変換マトリックス

例えば四角形Aにx方向に2倍，y方向に4倍の拡大を行い，さらに45°の回転を与え，（10，5）の平行移動を行ったイメージをA′とする。

〔図3－14〕



四角形A



四角形A′

この場合，変換マトリックスを用いて次のように表現することができる。

A′≪A．TRS．（10，5，45，2，4）……(1)

尚ここでグラフィック・オペレータ≪はイメージA′への代入操作を表わす。

また別の表現法として

A′≪A．TRS．（，，，2，4）．TRS．（，，45）．TRS．(10,5)

……(2)

と表わすこともできる。

2次元変換マトリックスの一般形は次のような形をとる。

（△x，△y，θ，Sx，Sy）

ここで各パラメータは

△x　：　x方向への平行移動量

△y　：　y方向への平行移動量

θ　：　原点を中心にx軸からy軸への回転角（単位は度）

Sx　：　x方向への拡大率

Sy　：　y方向への拡大率

である。内部的な処理としては，ｘ方向，ｙ方向への拡大→原点中心の回転→ｘ方向，ｙ方向への平行移動という順に変換が行われるようにマトリックスの演算を行っている。従ってこれと異なる順に変換したい場合は変換操作を分解し，任意の順序に変換マトリックスを作用させれば良い。左から順に変換処理がほどこされる。２次元座標変換マトリックスでは５つのパラメータを必要とするが，不要なパラメータは省略するかまたは▼，▼でおきかえることができる。

例えばｘ方向の平行移動のみを表わす変換マトリックスは（ｘ）と表現でき，内部的には（ｘ，０，０，１，１）と解釈される。又回転のみを表わす変換マトリックスは（，，θ）と表現すればよい。

(2)　３次元座標変換マトリックス

３次元での拡大，回転，平行移動を表わす３次元座標変換マトリックスは，２次元座標変換マトリックスと同様に次のような一般形をとる。

（△ｘ，△ｙ，△ｚ，θｘｙ，θｚ，Ｓｘ，Ｓｙ，Ｓｚ）

ここで各パラメータは，

△ｘ　：　ｘ方向への平行移動量

△ｙ　：　ｙ方向への平行移動量

△ｚ　：　ｚ方向への平行移動量

θｘｙ　：　原点を中心とするｘ－ｙ平面での回軸角

θｚ　：　原点を中心としたｘ－ｙ平面からｚ軸への回転角

Ｓｘ　：　ｘ方向への拡大率

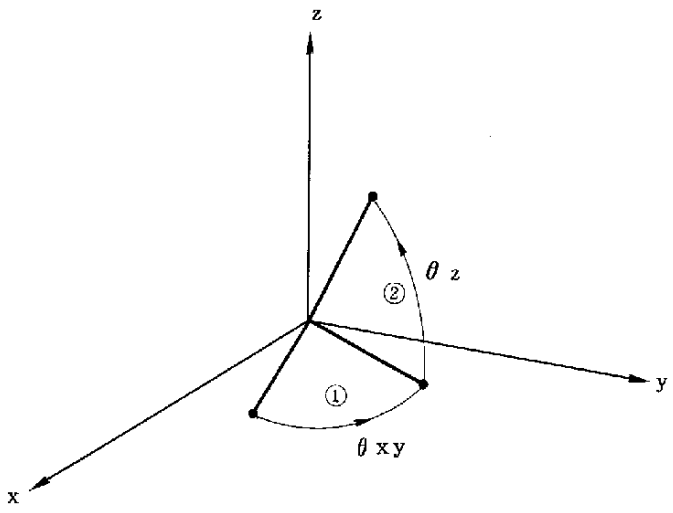
Ｓｙ　：　ｙ方向への拡大率

Ｓｚ　：　ｚ方向への拡大率

である。

変換の順序は〔図３－１５〕に示すように

〔図3－15〕



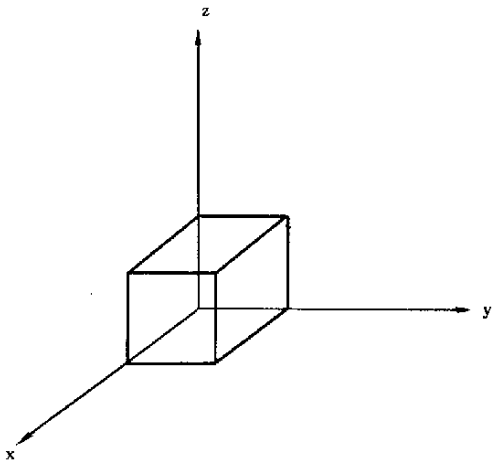
x−y平面に平行な面上でｘ軸からｙ軸へ，角度θｘｙの回転を行った後，その面からｚ軸へ，角度θｚの回転を行う。

例えば〔図３－１５〕の立方体イメージＡを，ｘ，ｙ，ｚ各方向に0.5，2.0，0.5倍し，ｘ－ｙ平面に関して４５°の回転を施し，さらに（１０，１０，１０）の平行移動の変換を行ったイメージをＡ′とすれば

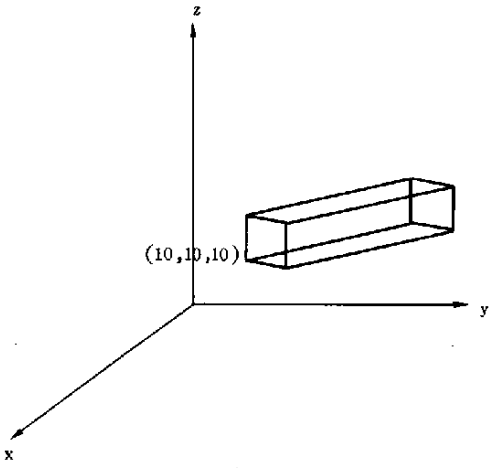
Ａ′←Ａ．ＴＲＳ３．（１０，１０，１０，４５，０，0.5，２，0.5）

と表現することができる。

〔図3－16〕



イメージA



イメージA'

(3) 透視変換マトリックス

3次元イメージを透視するには透視変換マトリックスをグラフィック・オペレータ.PERS.によってイメージに作用させる。透視変換マトリックスの一般形は次のような形をとる。

（F，x_0，y_0，z_0）

ここで各パラメータは

F　：　投象面を表わす

＝　1の時　　x－y平面

＝　2の時　　y－z平面

＝　3の時　　z－x平面

x_0　：　視点のx座標

y_0　：　視点のy座標

z_0　：　視点のz座標

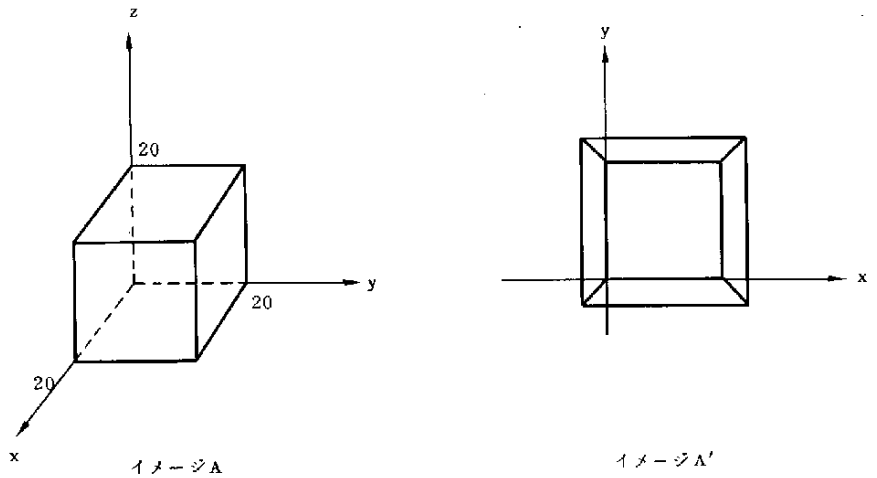
である。

透視を行う場合，投象面をどのように定めるかが問題となるが，投象面をx－y平面，y－z平面，z－x平面に限ることにより，マトリックスで表現することが可能となる。

次に〔図3－17〕に透視変換の例を示す。

立方体のイメージAに（100，10，10）を視点とし，x－y平面に透視した図形をイメージA′とする。

〔図3－17〕



イメージA　　イメージA′

(4) 平行投影マトリックス

3次元イメージの平面図，側面図，立面図の三面図を得るには，グラフィック・オペレータ.ORTH.によって平行投影マトリックスをイメージに作用させる。

平行投影マトリックスは次のような一般形をとる。

（F）

F ： 投影面を表わす。

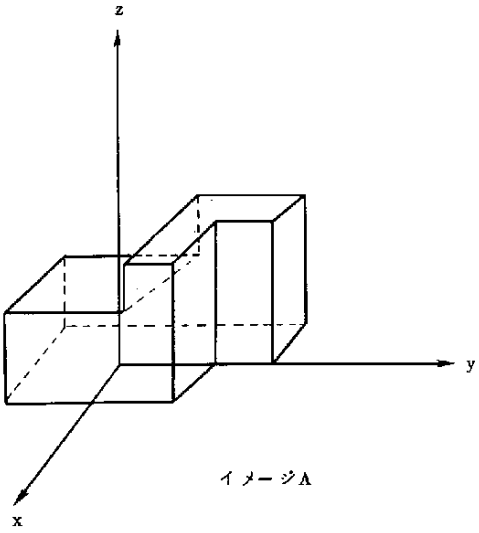
= 1の時 x－y平面

= 2の時 y－z平面

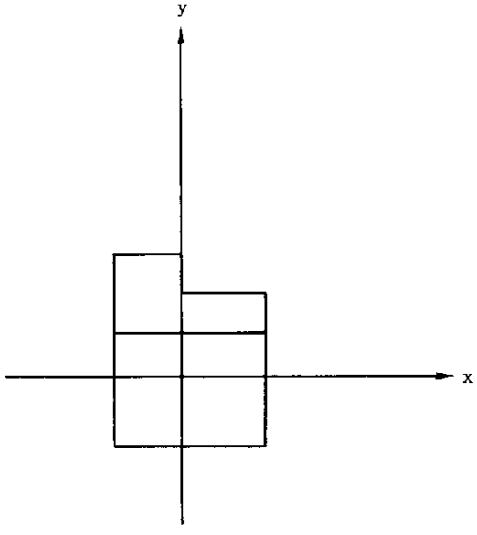
= 3の時 z－x平面

〔図3－18〕に平行投影の例を示す。イメージAをx－y平面に平行投影したイメージをA′，y－z平面に平行投影したイメージをA″とする。

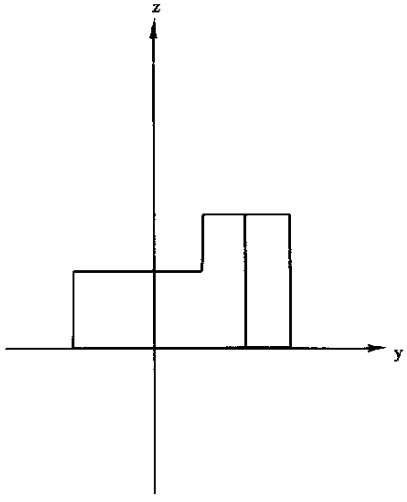
〔図3－18〕



イメージA



イメージA′



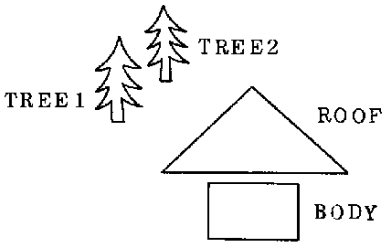
イメージA″

3.2.3 図 形 併 合

前の項では，図形に座標変換を行うことについて述べたが，複雑な図形を表現するには，さらに図形の併合操作も必要となる。イメージAとイメージBが併合される時，グラフィック・オペレータ＋によってA＋Bと表現される。

例えば〔図３－１９〕をイメージＰＩＣＴＲＥとすると次のように表現される。

〔図3－１９〕



HOUSE＜＝ＲＯＯＦ+ＢＯＤＹ

ＰＩＣＴＲＥ＜=HOUSE+TREE1
+TREE2

ここではイメージＲＯＯＦとイメージＢＯＤＹを併合してイメージＨＯＵＳＥを定義している。さらにイメージＨＯＵＳＥとＴＲＥＥ１，ＴＲＥＥ２を併合しイメージＰＩＣＴＲＥを形成している。このようにイメージは階層的に表現することが可能である。

3.2.4 図形のグルーピング

イメージはディスプレィ画面に表示され，そのイメージに対し何らかのアクションが与えられる場合，その識別単位となる論理的なまとまりをエンティティという。エンティティとイメージの対応をつける方法の一つとして階層的に定義されるイメージの最下層のイメージをエンティティとする考え方がある。この場合，ユーザはイメージとエンティティの対応を常に意識していなければならない。もう一つの方法として，図形出力文で指定されたイメージ単位を常にエンティティとする方法がある。しかしこの場合も，階層的にイメージを定義して

おきながら，出力時に，識別できるイメージの単位を考慮しなければならない。そこでUNGLでは，論理化因子というものを設け，イメージを定義していく段階で，論理的なまとまりを定義していく方法をとっている。

論理化因子は，エンティティの識別名と，ディスプレイ画面に表示される時の制御情報をもっている。識別名は整数，実数，ホラリス・コンスタント（Hollerith constant）のいずれをとることも可能である。またデータ・ストラクチャの要素であるアイテムを指定することもできる。制御情報はライトペン・ディテクト（light pen ditect）の可否と輝度を指定する情報を保持している。

論理化因子は＜論理名，制御情報＞という形式をとり，グラフィック・オペレータ｜によってイメージに付加される。

例えば〔図３－１９〕のイメージPICTREを論理化因子によって，次のようなグルーピングをすると

① PICTRE<=(HOUSE+TREE1+TREE2)｜ <1>

② PICTRE<=HOUSE｜ <1>+(TREE1+TREE2)｜ <2>

③ PICTRE<=ROOF｜ <1>+BODY｜ <2>+TREE1｜ <3>
+TREE2｜ <4>

①の場合には，HOUSE，TREE1，TREE2のどのイメージがライトペンでピックされても▼1▼のレスポンスが返ってくる。即ちどこをピックしてもPICTREという１つの画面をピックしたものとしか識別しない。②の場合にはHOUSEならば▼1▼，TREE1，TREE2ならば▼2▼のレスポンスが返ってくる。即ちHOUSEをピックすればHOUSEのみが対象となりTREEは全く無関係である。また③ではROOF，BODY，TREE1，TREE2のそれぞれのイメージに別々にアクションを加えることが可能である。

論理名は，この例のように整定数であってもよいが，次のようにアイテムを論理名とすることもできる。

PICTRE<=ROOF｜<IROOF>+BODY｜<IBOBY>+TREE$_1$｜
<ITREE$_1$>+TREE$_2$｜<ITREE$_2$>

PUT （ITREE1） IN TREES

PUT （ITREE2） IN TREES

ここでIROOF，IBODY，ITREE1，ITREE2はアイテムであり，相互の関連を，この例の様にPUTステートメントで表現することができる。ここではアイテムITREE1とITREE2をセットTREESのメンバとしている。

出力された図形に何のアクションも加える必要がなく，単にイメージをディスプレイ画面に表示することを目的とする場合には論理化因子を付加する必要はない。

3.2.5 図形表現

これまで述べてきたように，イメージは単にイメージ・データによって定義されるのではなく，それらのイメージに図形変換，図形併合操作，グルーピングを行うことによって更に複雑な図形を表現することができる。

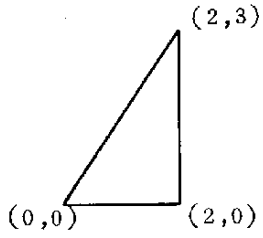
すなわちUNGLでは複雑な図形もイメージ・データ，イメージ，変換マトリックス，論理化因子，グラフィック・オペレータを組み合わせて用いることにより容易に表現できる。そしてこの表現形式をイメージ・エクスプレション（image expression）と呼ぶ。

次にイメージ・エクスプレションの例をあげる。

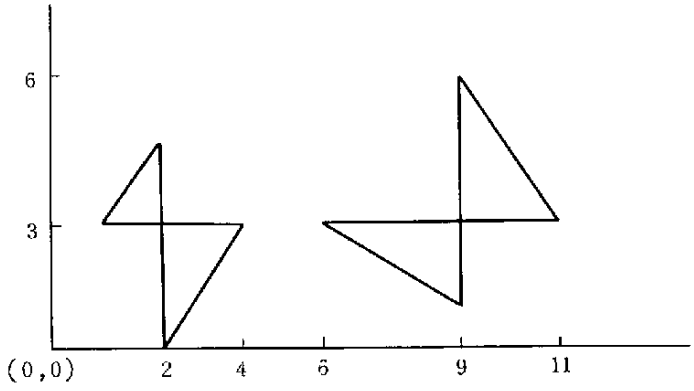
一つの三角形イメージAから〔図３－２０〕で示すようなイメージBを定義するには，次のように表現できる。

```
B<=A.TRS.(1,3,0,0.5,0.5)
    +A.TRS.(4,3,180)
    +A.TRS.(9,6,-90,1.5,0.667)
    +A.TRS.(9,1,90)
```

〔図3－20〕



イメージA



イメージB

3.3 イメージ・データ・ストラクチャ

3.3.1 イメージ・データ・ストラクチャの概要

イメージは定義されると内部的にはデータ・ストラクチャとして表現される。すなわち図形記述ステートメントは実際にはデータ・ストラクチャの記述であり，実行時に内部にデータ・ストラクチャが作成される。このようにイメージの実体はイメージ・データ・ストラクチャとして存在する。またイメージが再定義されるということは，このストラクチャに修正を加えることを意味する。しかしユーザはデータ・ストラクチャを意識する必要はない。

イメージの出力ステートメントDRAWあるいはANIMATEが実行されると，このデータ・ストラクチャをトレイスし画面に表示するイメージのデータを抜き出す。このように出力ステートメントが出される度に，構造をトレイスし，データを作成するということは処理時間に問題があるが，構造化したことにより動的な図形操作を可能にしている。例えば次のようなステートメントを考えてみる。

```
      CIRCLS <=CIRCL.TRS.(,,,R,R)
      DO 100 I=1,10
      DRAW  0,CIRCLS
      R=R+10
  100 CONTINUE
```

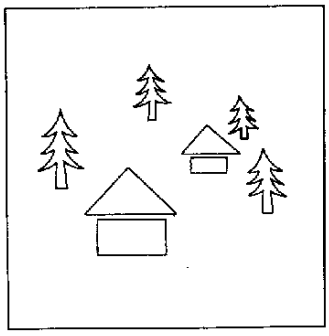
ここでCIRCLは円を表わすイメージとすると，DO以下のステートメントが実行されることにより，ディスプレイ画面には半径が１０づつ大きくなった円が１０描かれる。この時CIRCLSの構造は変わらない。

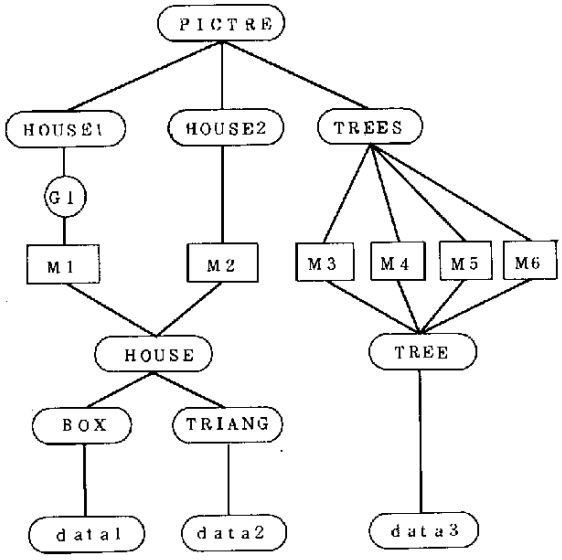
イメージは階層をもち，出力時のストラクチャのトレイスは下方向へのトレイスのみであることから，トリー構造を採用した。さらに，一度定義されたイメージは，他のイメージを定義するために幾度でも利用することができるので，実際にはクロス・コネクションを許すトリー構造になっている。例えば〔図３－２１〕のような図形構造は次のようなステートメントで記述さ

れる。

```
TREE <= ▼LINE(……data1……)▼
TRIANG <= ▼LINE(……data2……)▼
BOX <= ▼LINE(……data3……)▼
HOUSE <= TRIANG + BOX
HOUSE1 <= HOUSE.TRS.(…M1…)  <G1>
HOUSE2 <= HOUSE.TRS.(…M2…)
TREES<= TREE.TRS.(…M3…)+TREE.TRS.(…M4…)
        +TREE.TRS.(…M5…)+TREE.TRS.(…M6…)
PICTRE <= HOUSE1+HOUSE2+TREES
```

〔図3－21〕





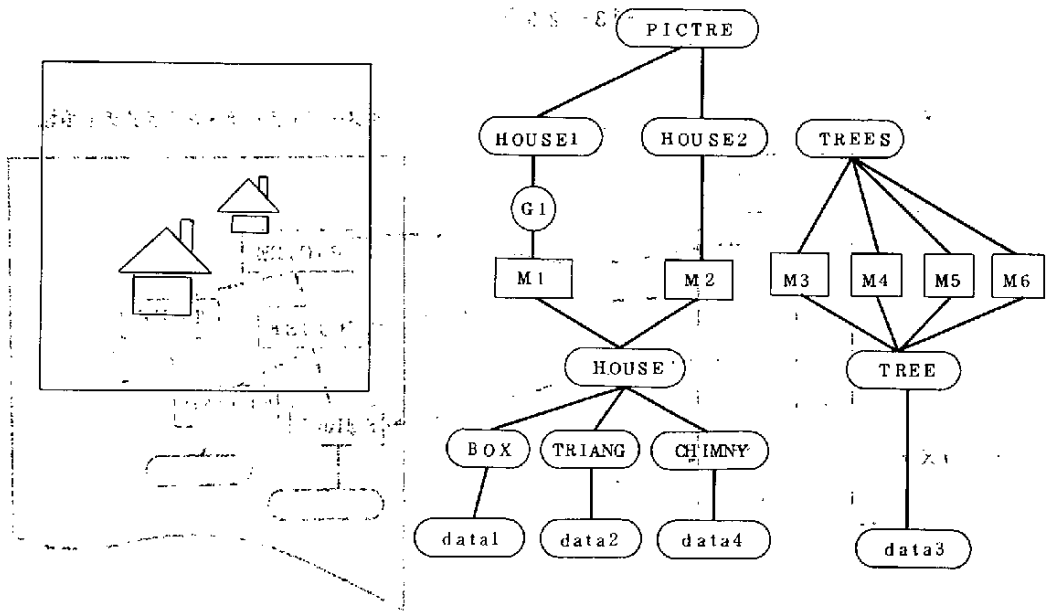
ここで次のようなステートメントが実行されると，イメージの構造は〔図3－22〕のように修正される。

CHIMNY <= LINE (…data4…)

HOUSE <= HOUSE +CHIMNY

PICTRE <= HOUSE1 + HOUSE2

〔図3－22〕

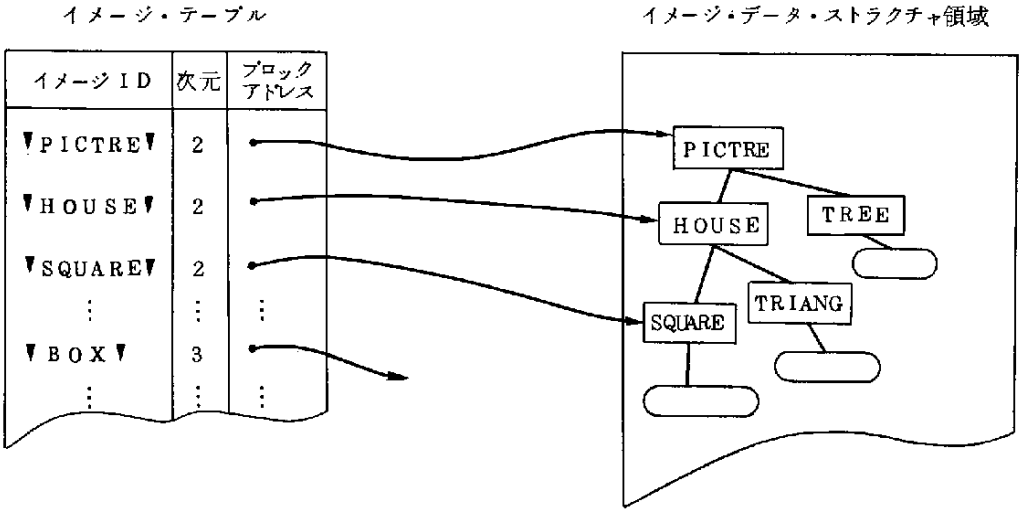


3.3.2 構造の要素

A. イメージ・テーブル

イメージが定義されると，そのイメージＩＤ，そのイメージの次元，対応するブロックのアドレスがイメージ・テーブルに登録される。イメージの構造を参照する時には，このテーブルを検索することによりブロック・アドレスからイメージ・データ・ストラクチャ領域内のイメージ所在を知ることができる。

〔図3－23〕



B. ブロック

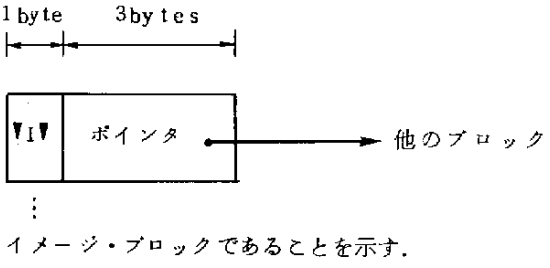
イメージ・データ・ストラクチャはイメージ・ブロック，マトリックス・ブロック，論理化因子ブロック，データ・ブロック，アソシェータ・ブロックの５種の要素から構成される。

(1) イメージ・ブロック

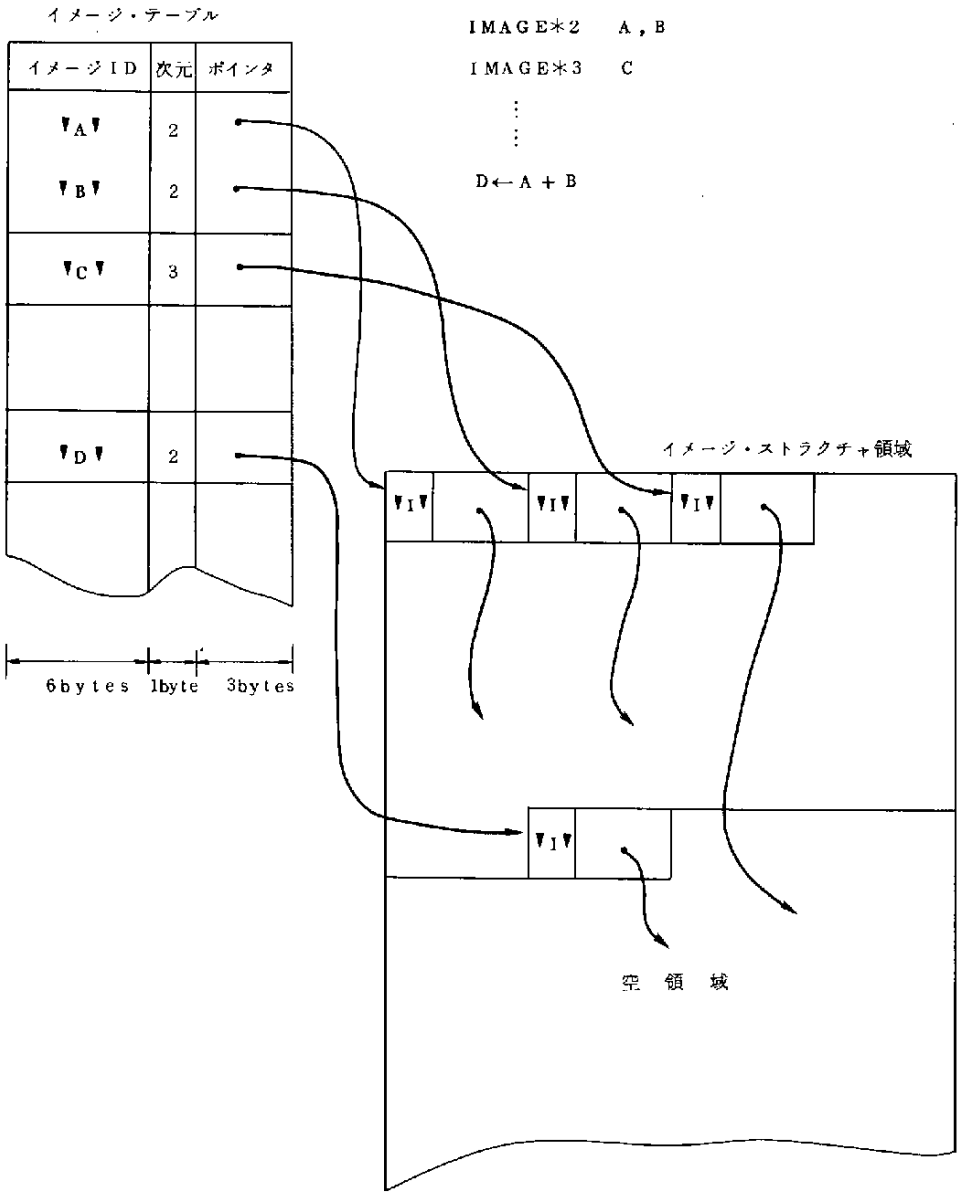
イメージ宣言文によって宣言されたイメージは，イメージ・テーブルに登録され，そのイメージのためにイメージ・ブロックが一つ創成される。又イメージ代入文で宣言されていないイメージが使われた時はその時点でブロックが創成される。

イメージ・ブロックは次の形式をとる。

〔図3－24〕

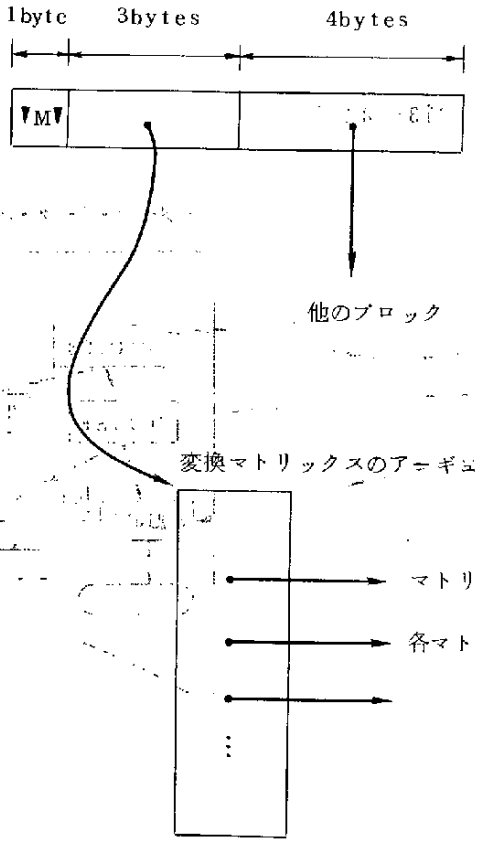


〔図3－25〕


イメージ・テーブル
イメージID
次元
ポインタ
▼A▼
2
▼B▼
2
▼C▼
3
▼D▼
2
6bytes
1byte
3bytes
IMAGE*2 A,B
IMAGE*3 C
D←A＋B
イメージ・ストラクチャ領域
▼I▼
▼I▼
▼I▼
▼I▼
空 領 域

(2) マトリックス・ブロック

イメージに，グラフィック・オペレータ．TRS．，．PERS．，．ORTH．で変換マトリックスが付加された時マトリックス・ブロックが創成される。



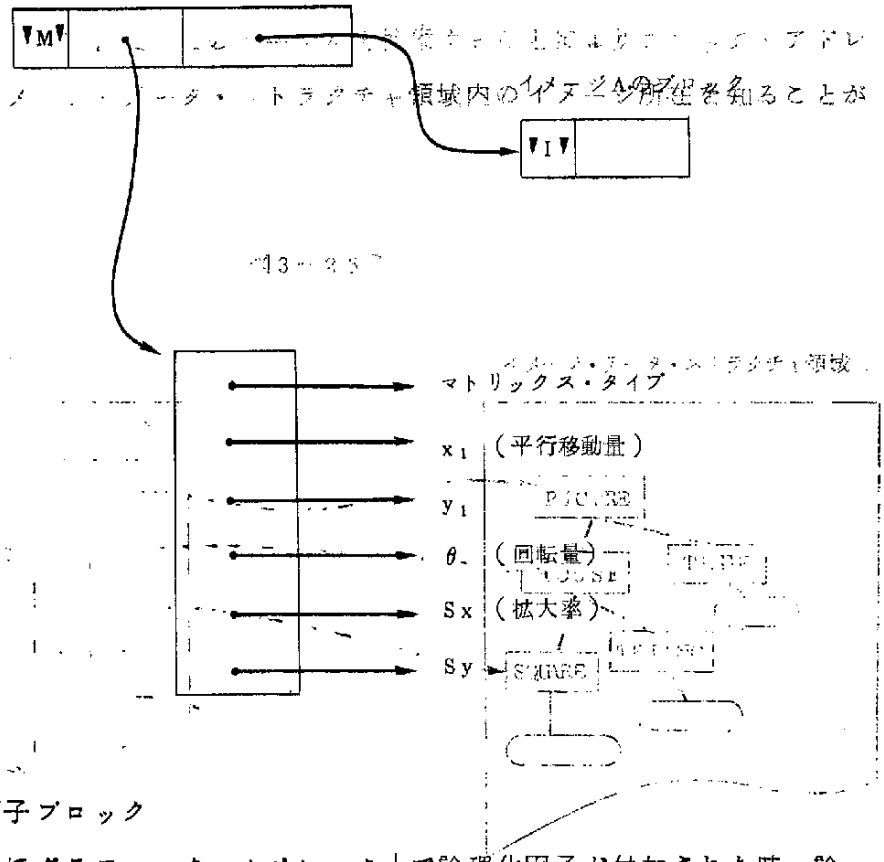
ここでアーギュメント・リストとはSYARGLによってFORTRANコンパイル時に作成されるアーギュメント・リストを示している。

例えば次のようにイメージAに座標変換を施すと，

A．TRS．（x_1，y_1，θ，Sx，Sy）

ストラクチャは〔図３－２７〕のように形成される。

〔図3－２７〕

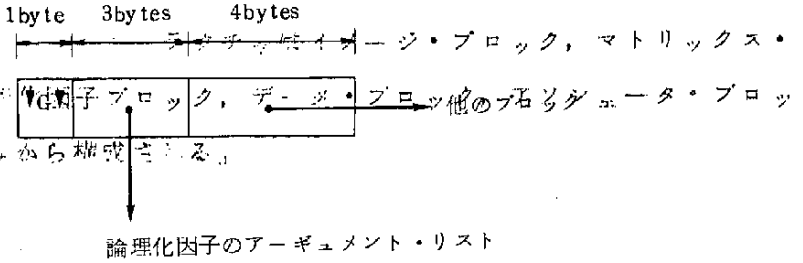


(3) 論理化因子ブロック

イメージにグラフィック・オペレータで論理化因子が付加された時，論理化因子ブロックが創成される。

論理化因子ブロックは次の形式をとる。

〔図3－２８〕



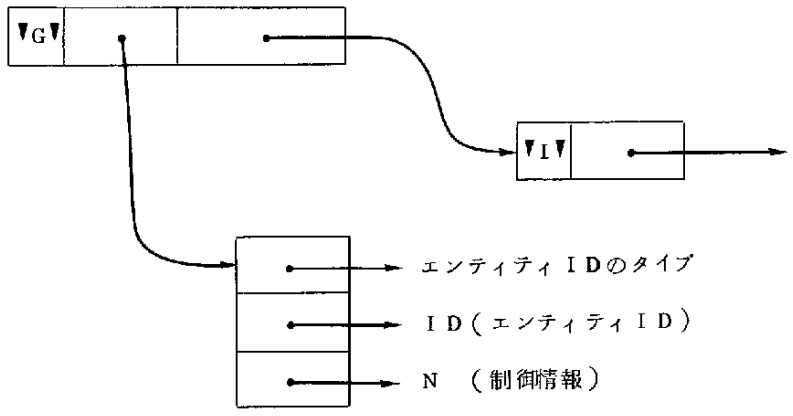
論理化因子のアーギュメント・リスト

例えば次のようにイメージAに論理化因子を付加すると，

A | ＜ＩＤ，Ｎ＞

ストラクチャは〔図３－２９〕のように形成される。

〔図3－２９〕

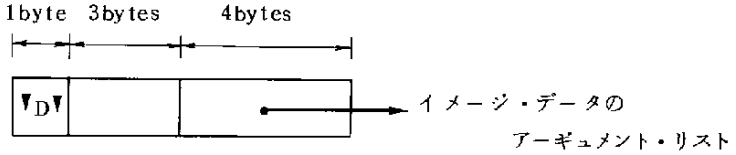


(4) データ・ブロック

イメージ・データが定義された時に創成され，イメージ・データの実値を示すアーギュメントリストをリンクする時のヘッドとなる。

データ・ブロックは次の形式をとる。

〔図3－３０〕



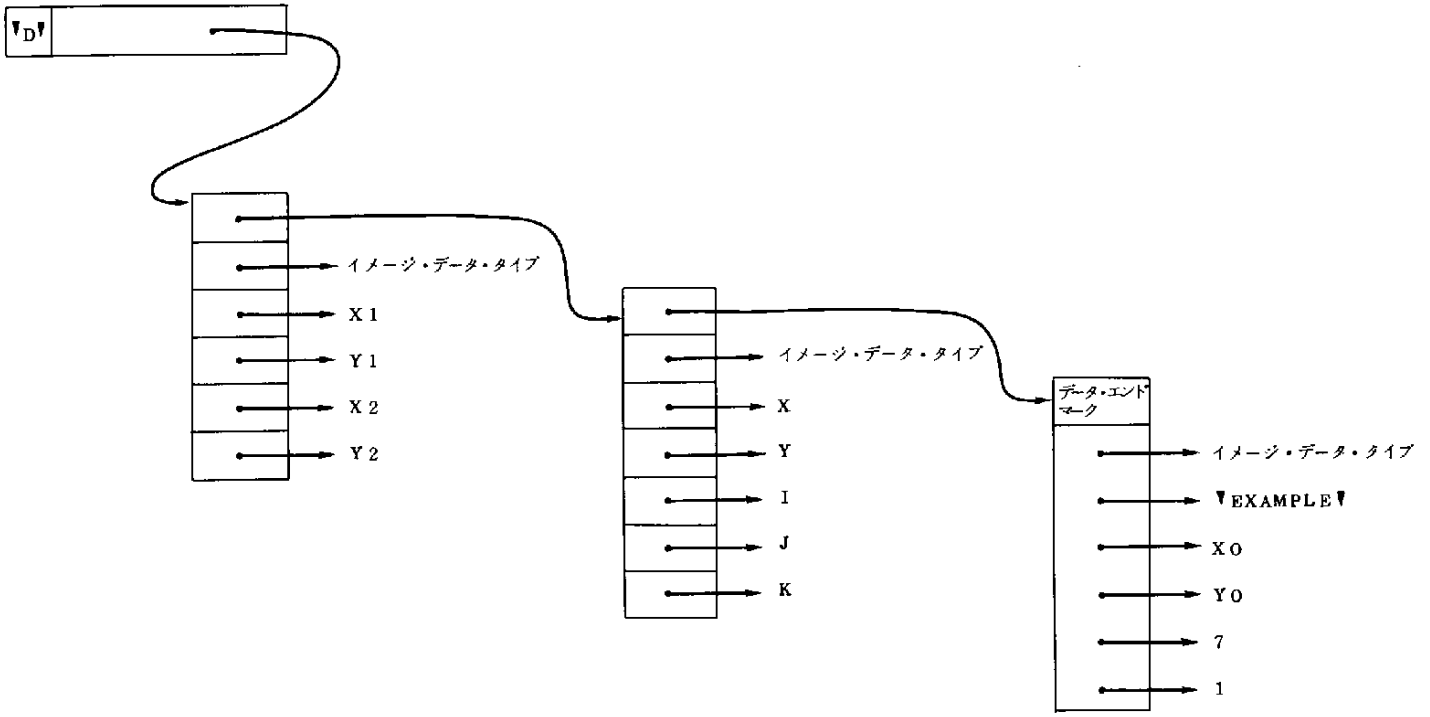
例えば次のようなイメージ・データに対して

▼ＬＩＮＥ（Ｘ1，Ｙ1，Ｘ2，Ｙ2），ＬＩＮＥ（Ｘ，Ｙ，Ｉ，Ｊ，Ｋ），

ＴＥＸＴ（▼ＥＸＡＭＰＬＥ▼，ＸＯ，ＹＯ，７，１）▼

ストラクチャは〔図３－３１〕のように形成される。

〔図 3－31〕

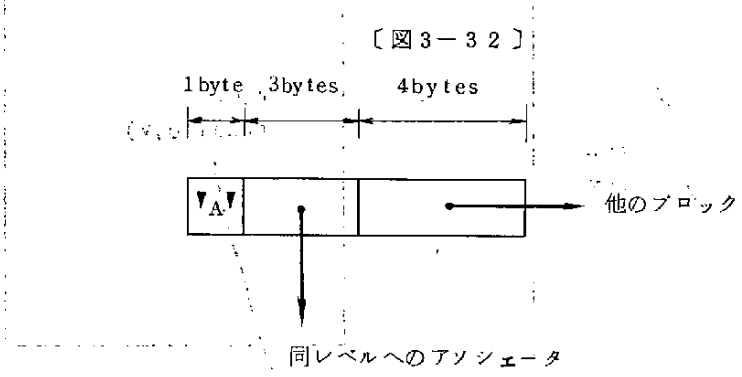

'D'
イメージ・データ・タイプ
X1
Y1
X2
Y2
イメージ・データ・タイプ
X
Y
I
J
K
データ・エンド・マーク
イメージ・データ・タイプ
'EXAMPLE'
XO
YO
7
1

(5) アソシェータ・ブロック

グラフィック・オペレータ＋によってイメージの併合が行われる時，アソシェータ・ブロックが創成される。

アソシェータ・ブロックは次の形式をとる。

〔図3－32〕

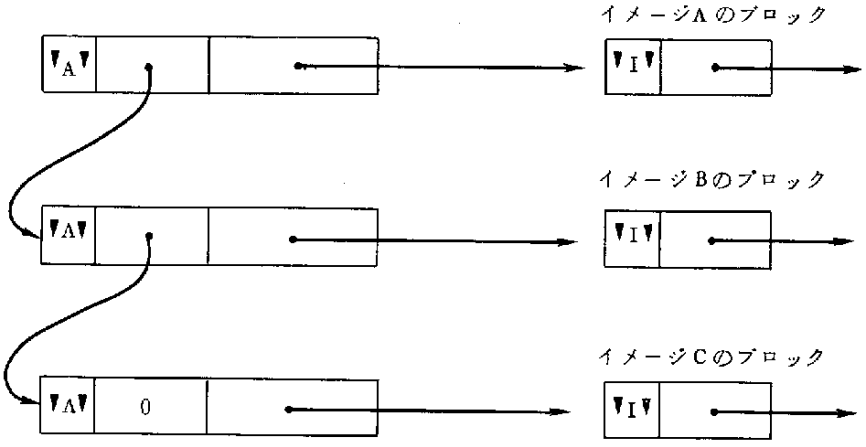


例えば次のようにイメージAとイメージBとイメージCを併合すると

A ＋ B ＋ C

ストラクチャは次のように形成される。

〔図3－33〕



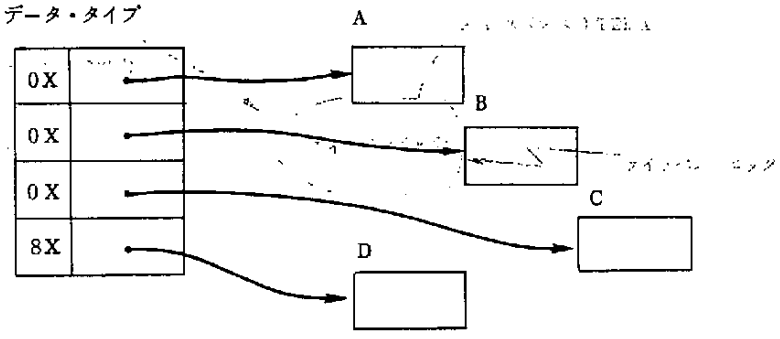
3.3.3 構造化と修正

A．図形の構造化

これらのブロックは，イメージ代入文が実行されることにより，イメージ・データ・ストラクチャ領域に構造化される。イメージ代入文はプリコンパイル時に，リバースポリシュ・ノーティションにおとされ，それに対するオブジェクトが生成される。オブジェクトは全てサブルーチン・コールの形がとられ，FORTRANが作成するアーギュメントリストを実行時に構造に組み込む。すなわち，次のようなステートメントからFORTRANは〔図３－３４〕のようなアーギュメント・リストを作成する。

```
CALL SUB (A, B, C, D)
```

〔図3－34〕



さらに実行時には，このアーギュメント・リストの先頭アドレスが知らされ，各ブロックのポインタがこのアドレスを指すことができる。

次に，オブジェクトが実行されることにより，実際にどのように構造化されていくかを述べる。

例えばイメージ代入文

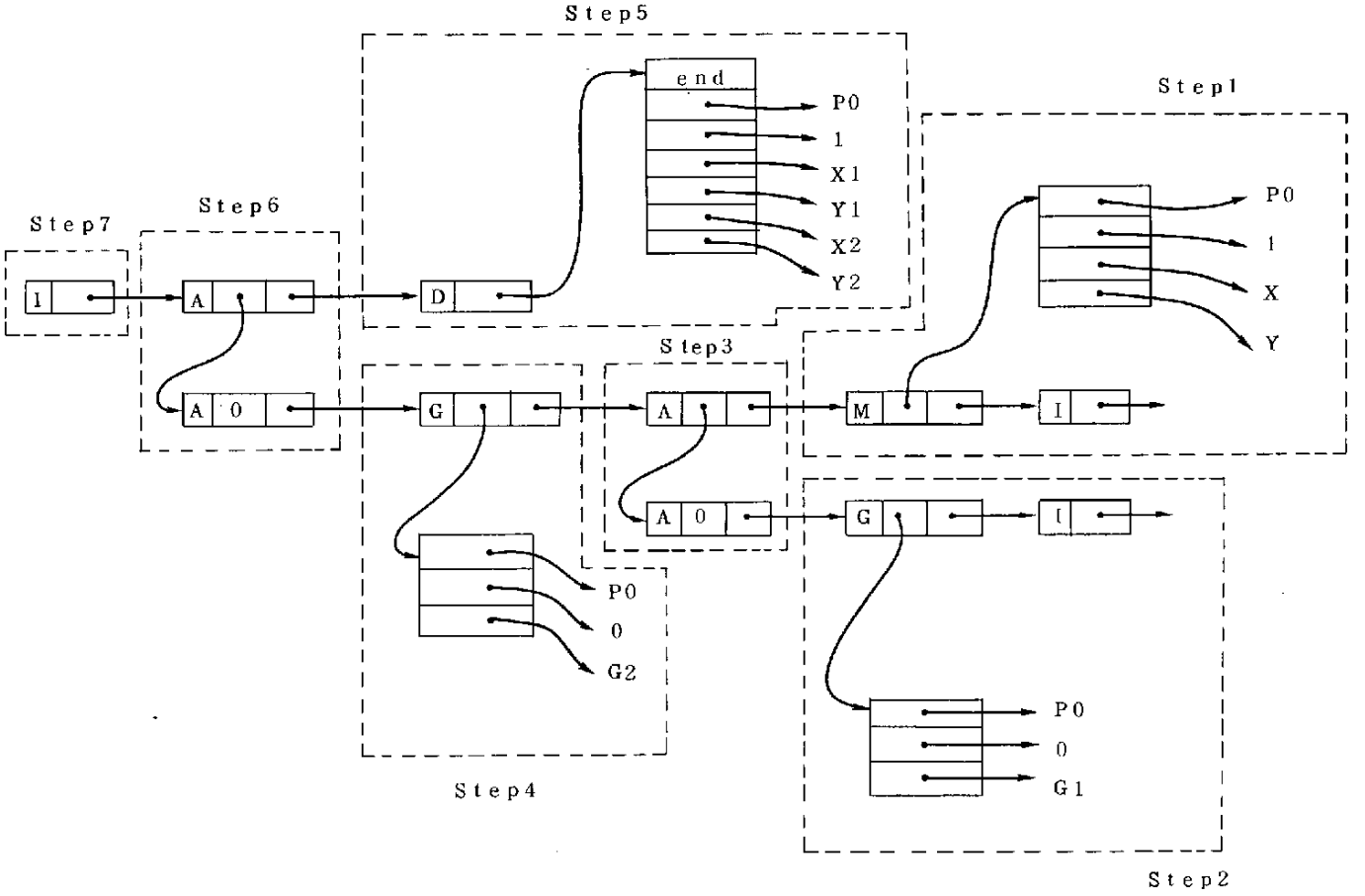
```
A<=(B.TRS.(X,Y)+C|<G1>)|<G2>+
                        'LINE(X1,Y1,X2,Y2)'
```

に対し次のようなオブジェクトが作成される。

```
CALL  S¥ARGL    (P0, 1, X, Y)                          ……step1
CALL  S¥MATX    (P1, 'B␣␣␣␣␣', 0, P0)

CALL  S¥ARGL    (P0, 0, G1)                            ……step2
CALL  S¥GFAC    (P2, 'C␣␣␣␣␣', 0, P0)

CALL  S¥IMAD    (P3, P1, 0, P2, 0)                     ……step3

CALL  S¥ARGL    (P0, 0, G2)                            ……step4
CALL  S¥GFAC    (P4, P3, 0, P0)

CALL  S¥ARGL    (D, P0, 1, X1, Y1, X2,Y2)              ……step5
CALL  S¥IMDT    (P5, P0)

CALL  S¥IMAD    (P6, P4, 0, P5, 0)                     ……step6

CALL  S¥ASGN    ('A␣␣␣␣␣', P6)                         ……step7
```

これらのオブジェクトが実行されることにより〔図３－３５〕に示すような構造が作られることとなる。

〔図3－35〕


Step5
end
P0
1
X1
Y1
X2
Y2
Step1
P0
1
X
Y
Step7
Step6
I
A
A
0
D
Step3
G
A
M
I
A
0
G
I
P0
0
G2
P0
0
G1
Step4
Step2

次にその過程をステップ毎に追ってみる。

step1　S¥ARGLルーチンでアーギュメント・リストが作られ，そのアドレスがP0に入れられる。S¥MATXではイメージID'B␣␣␣␣␣'をイメージ・テーブルで検索し，ブロック・アドレスを知る。さらにマトリックス・ブロックを創成し，アーギュメント・リストへのポインタとイメージ・ブロックへのポインタをセットする。さらにマトリックス・ブロックのアドレスをP1に格納する。

step2　論理化因子ブロックを創成し，step 1と同様にポインタをセットし，論理化因子ブロックのアドレスをP2に格納する。

step3　P1，P2に格納されているアドレスのブロックがアソシェータ・ブロックでないため，アソシェータ・ブロックを二つ創成し各ブロックをリンクする。P3にはリンクされたブロックのヘッドとなるブロックのアドレスを格納する。もし格納されているアドレスのブロックがアソシェータ・ブロックであるならばアソシェータ・ブロックを創成する必要はない。

step4　step 2と同様

step5　S¥ARGLルーチンで作成されたアーギュメント・リストをS¥IMDTルーチンでリンクする。リンクするためのポインタはS¥ARGLのアーギュメントDとしてとられた領域を用いる。データ・ブロックを創成し，そのブロックをヘッドとしアーギュメント・リストをリンクする。P5にはデータ・ブロックのアドレスを格納する。

step6　step 3と同様

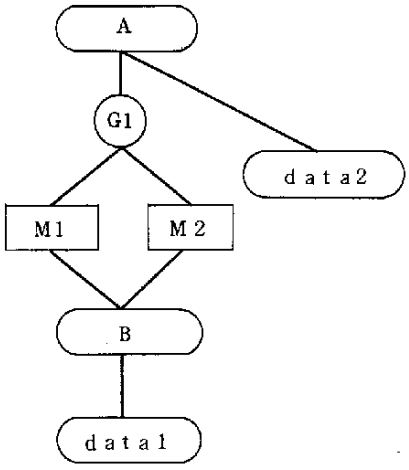
step7　S¥ASGNのアーギュメント'A␣␣␣␣␣'をイメージ・テーブルで検索し，ブロック・アドレスを知り，P6に格納されているアドレスのブロックをリンクする。

このようにしてイメージ代入文によって定義されたイメージは構造化される。

B．図形の修正

構造化されたイメージは再定義されると，構造の修正が行われる。例えばイメージAが〔図３－３６〕のように定義されていたとする。

〔図3－36〕



B<＝▼…data1…▼

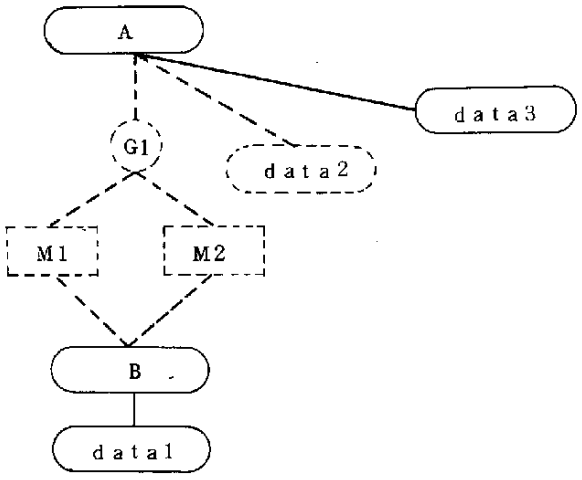
A<＝（B．TRS．(…M1…)

＋B．TRS．(…M2…))

｜<G1>+▼…data2…▼

この時，イメージAをA<＝▼…data3…▼のように再定義すると〔図３－３７〕のような構造となる。

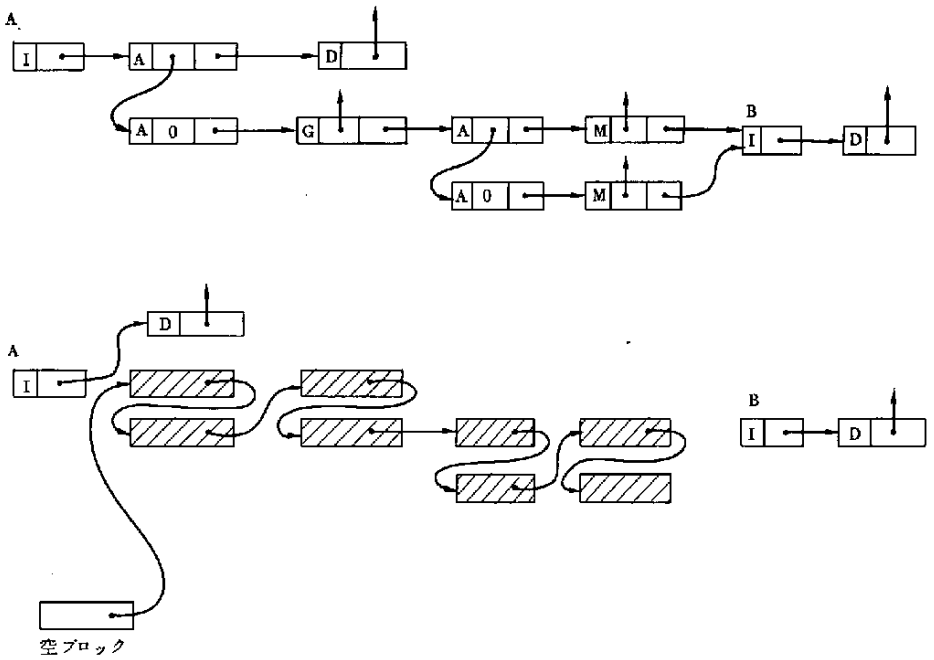
〔図3－37〕



Ｇ１，ｄａｔａ ２,Ｍ１，Ｍ２のデータはアクセスされず不必要となるため，各ブロックは空ブロックとされる。このように最上層にイメージ・ブロックを持たないブロックは意味を持たない。

イメージの再定義が行われると，再定義されたイメージの構造をトレイスしていき，イメージ・ブロックに到達するまでのブロックを空ブロックとする。〔図３－３８〕は，データ・ストラクチャ領域内での空ブロックの様子を示したものである。

〔図3－３８〕



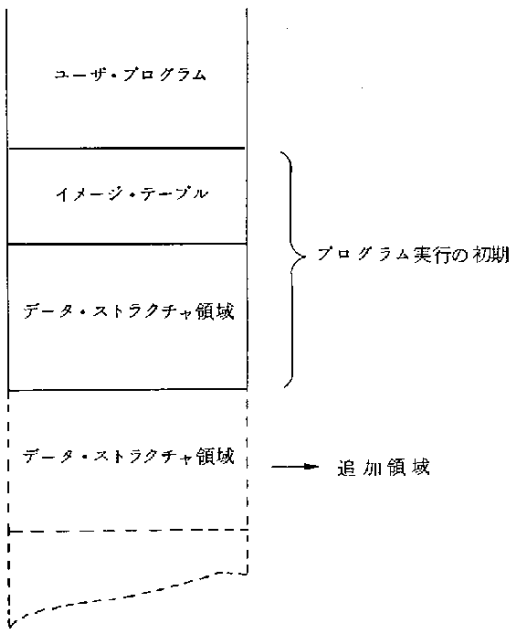
3.3.4 フリーストレッチの管理

イメージ・データ・ストラクチャの領域はコア内にとられる。2次記憶に拡張しなかったのは，次のような理由による。イメージの出力は指定された時点で構造をトレイスすることにより，表示データを作成し，ディスプレィ画面に表示する形をとる。この構造トレイスの過程では，マトリックスの演算，ウインドウイングの処理などが行われるので当然のことながらかなりの処理時間がかかることが予想される。そのため2次記憶へのアクセス，ページの管理などの処理が加わることは，よりいっそう表示までの時間を遅らせることとなる。インタラクティブ処理では，図形の出力の時間が大きな影響を与える。その点を重視しイメージ・データ・ストラクチャの領域はコア内にとることにした。そのためプログラムが終了するとストラクチャは消滅してしまう。このことは必要なイメージも構造化された形で残しておくことはできないことを意味し，一つの欠点となっている。

A．領域管理

イメージ・テーブルおよびデータ・ストラクチャ領域は，プログラム実行の初期にユーザ・プログラムの下に〔図3－39〕のようにとられる。

〔図3－39〕



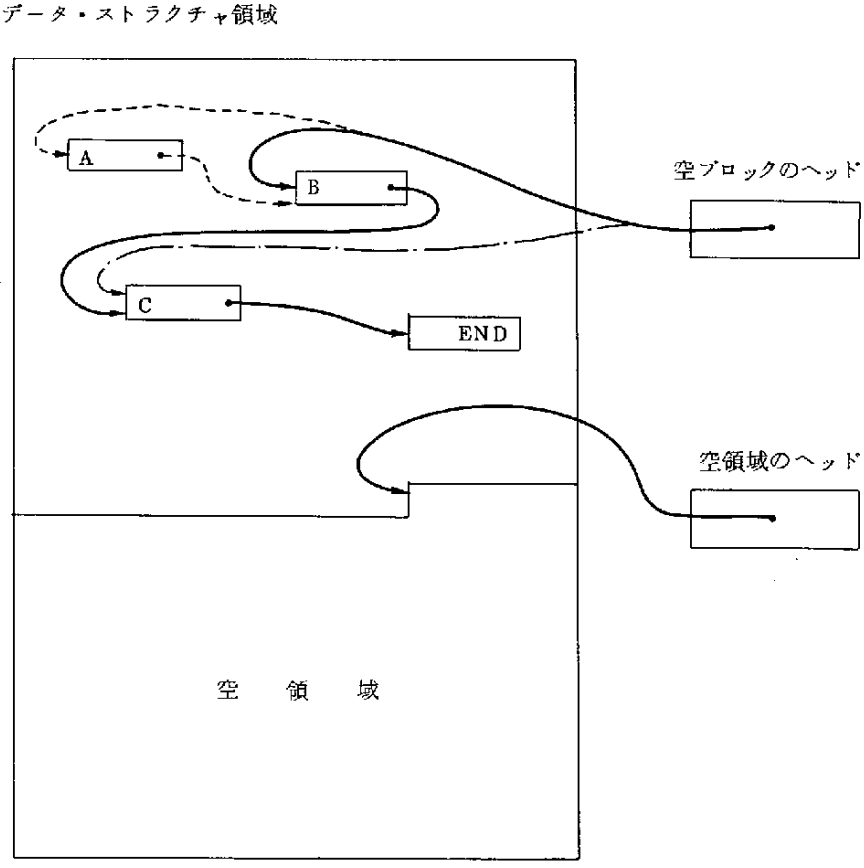
実行時にイメージ・データ・ストラクチャ領域が足りなくなった時には，さらにその下に，連続領域となるような空領域が動的に確保される。

データ・ストラクチャ領域では，ブロックによってイメージの構造化が行われるが，その時ブロックを創成したり，不要なブロックを空領域とするために，フリー・ストレッチ管理を行っている。

B．ブロック管理

イメージは一度定義されると修正されることはあっても消去されることはない。すなわちイメージ・ブロックは創成されるだけである。他のブロックは前述のように修正によって不要になることもある。またブロックは固定長である。フリー・ストレッチの管理を〔図３－４０〕に示す。

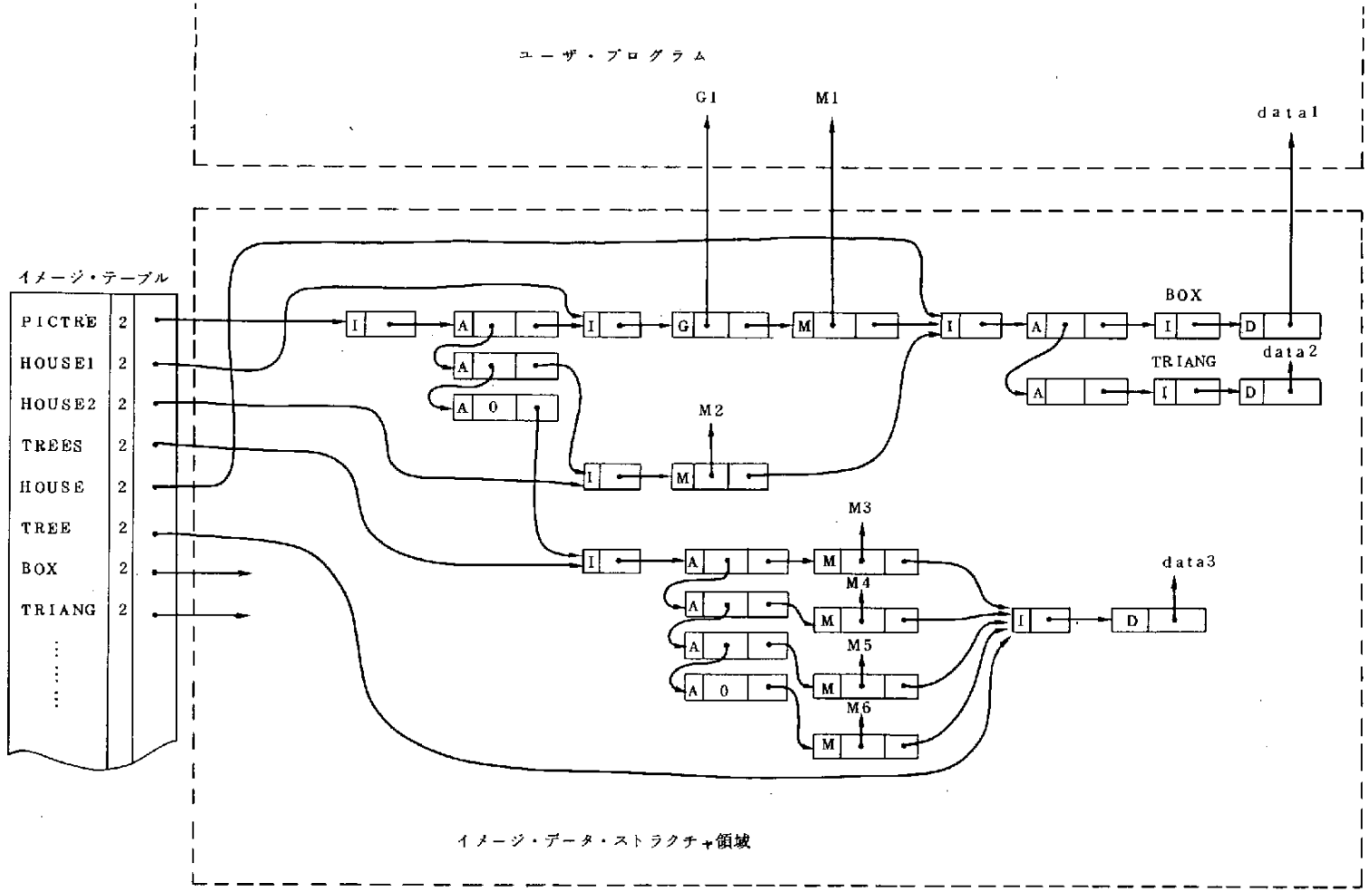
〔図3－40〕



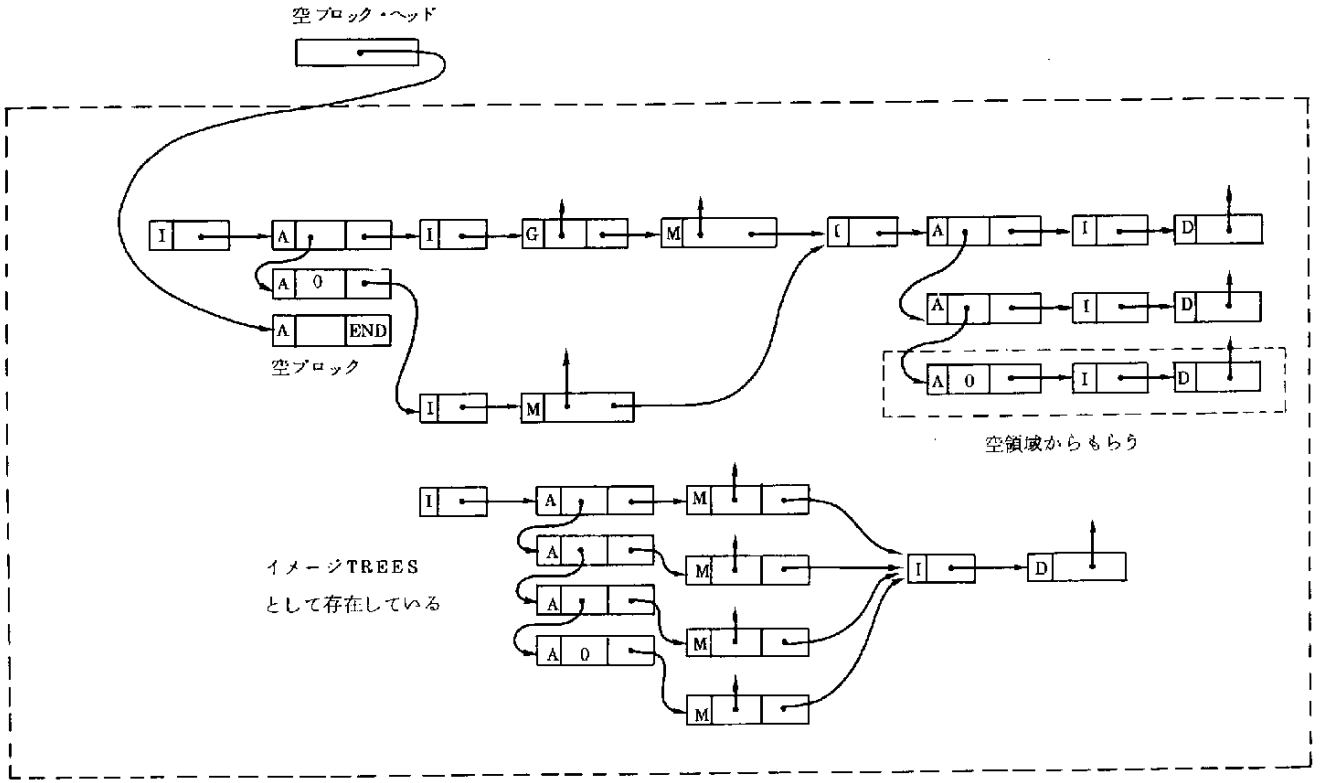
この図で示すようにブロックを１つ創成したい場合，空ブロックとしてリンクされているブロックＢが使われる。もし空ブロックがなければ空領域からとられ，さらに空領域がない場合には，今までの領域に連続した領域をとる。ブロックＡを空領域に返す時は，空ブロックのリンクに挿入する。

例えば〔図３－２１〕はイメージ・データ・ストラクチャ領域に〔図３－４１〕のように構造化され，〔図３－２２〕のような修正を施すことにより〔図３－４２〕のように構造は修正される。

〔図3－41〕


ユーザ・プログラム
G1
M1
data1
イメージ・テーブル
PICTRE 2
HOUSE1 2
HOUSE2 2
TREES 2
HOUSE 2
TREE 2
BOX 2
TRIANG 2
BOX
TRIANG
data2
M2
M3
M4
M5
M6
data3
イメージ・データ・ストラクチャ領域

〔図3－42〕


空ブロック・ヘッド
I
A
I
G
M
I
A
I
D
A
0
A
END
空ブロック
I
M
A
I
D
A
0
I
D
空領域からもらう
I
A
M
A
M
A
M
A
0
M
I
D
イメージTREES
として存在している

3.3.5 ストラクチャ間のインターフェイス

ここでは問題向データ・ストラクチャと図形データ・ストラクチャの関連に関して説明を加える。

GRINⅡ，AIDSにみられるように，図形データ・ストラクチャに問題向けデータも組み込むようなデータ構造を採用しているものもある。この方法によれば図形データと問題向データが一つのデータ・ストラクチャに組み込まれることによって図形データと問題向データは直接的に関連を記述することができ，インタラクションによる図形情報はそのまま問題向データに反映されるという特徴がある。しかし構造は非常に複雑なものとなり，同時にプログラムの記述性も悪くなる。

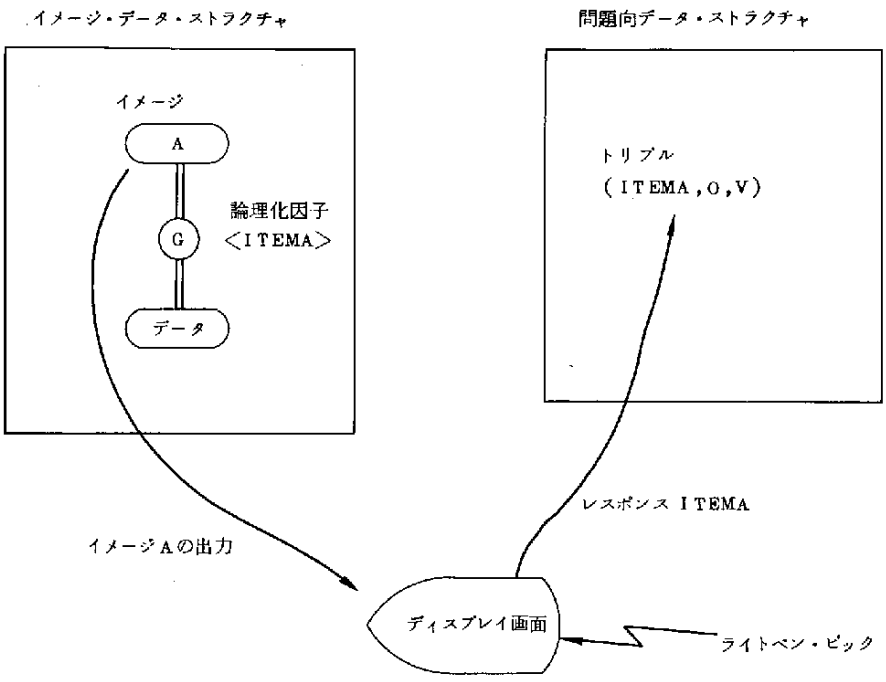
そこでUNGLでは二つの構造を分離して扱う方法をとっている。問題向データ・ストラクチャはオーダード・トリプルとして表現されるLEAPタイプのストラクチャとし，図形データ・ストラクチャは，クロス・コネクションを許すトリー構造としている。

この二つのストラクチャは独立に操作できるが，データのコミュニケーションは当然必要となる。そこでUNGLではイメージの構造に組み込まれる論理化因子の論理名として，データ構造のエレメントであるアイテムを指定することができるようにした。

例えばイメージAに論理名としてアイテムITEMAを付加すると，ディスプレイ画面に表示されたイメージAがライトペン・ピックされた時には，レスポンスとしてITEMAがプログラムに返される。

これによって問題向構造へのアクセスが可能になっている。〔図3－43〕にその様子を示す。

〔図３－４３〕


イメージ・データ・ストラクチャ
問題向データ・ストラクチャ
イメージ
A
G
データ
論理化因子
<ITEMA>
トリプル
(ITEMA,O,V)
レスポンス ITEMA
イメージAの出力
ディスプレイ画面
ライトペン・ピック

3.4 イメージの出力

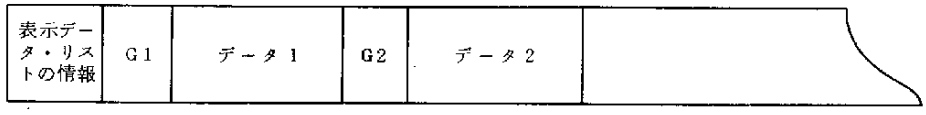
イメージ代入文によってイメージは構造化されるが，そのイメージが出力されるのはイメージ出力文がだされた時である。イメージ出力文が実行されると，指定されたイメージの構造が図形変換を行いながらトレイスされ，エンティティ単位にまとめられた表示用データの集りである表示データ・リストが作成される。

イメージ出力文にはDRAWステートメントとANIMATEステートメントがある。DRAWは単にイメージの表示を行うが，ANIMATEはイメージ単位に動きの異なるアニメーションを行うことができる。

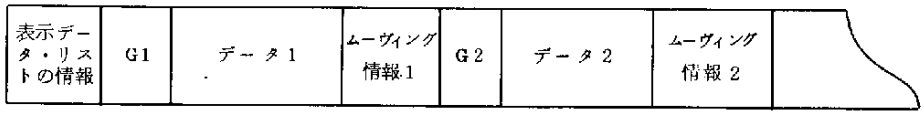
出力時にはウインドウイング情報を与え（２－２－３C参照），イメージの一部分だけを表示することもでき，実際に表示されるデータが編集されて〔図３－４４〕のような表示データ・リストが作られる。

〔図3－44〕

DRAW



ANIMATE



表示データ・リストの情報は，表示データ・リストの長さ，リスト中に含まれるエンティティの数を蓄えている。エンティティ情報はエンティティ・ネーム（論理名），エンティティ長，エンティティの制御情報を蓄えている。またデータは図形の最も基本となる要素である点と線，文字列を表わすエレメントの集まりである。さらにANIMATEの時には，アニメーションの動

きを表現しているムーヴィング情報がエンティティ単位に付けられる。この表示データ・リストの形式はDXCを経由してGCHに送られるデータ形式である。

3.4.1 イメージ構造のトレイス

イメージ出力文がだされると，システムはイメージの構造を次のようなステップを踏んでトレイスを行う。

まずカウンタとして次のものを設ける。

RANK　：ブロックの階層を示す。

MRANK　：各階層のブロックがマトリックスであるか否かを示す。（各階層毎にもつ）

GRANK　：直前に出された論理化因子の階層

MWORK　：現在，何番目のマトリックス演算領域を使用中かを示す。

NEXTAD　：同レベルの階層で次にリストされているアソシェータのアドレスを入れておく。リストされていない時は0とする。

step 1　イメージ・テーブルから指定されたイメージのブロック・アドレスを知る。

RANK	0
MRANK	すべてOFF
GRANK	0
MWORK	0
NEXTAD	すべて0

step 2　RANK ← RANK+1

次にリンクされているブロックの種類により各ステップに進む。

イメージ・ブロック	step 2 へ
アソシェータ・ブロック	step 3 へ
マトリックス・ブロック	step 4 へ
論理化因子・ブロック	step 5 へ
データ・ブロック	step 6 へ

step 3　同レベルの階層で次にリンクされているアソシエータ・ブロックのアドレスをNEXTAD(RANK)へ格納する。もし存在しなければNEXTAD(RANK)←0とする。

step 2へ

step 4　MWORK ＝ 0の時

変換マトリックスのデータを演算するために実際のマトリックスの値に変換してマトリックス演算ワーク領域へ格納する。

MWORK ≒ 0の時

変換マトリックスのデータをマトリックスの値に変換しMWORK番目のワーク領域のマトリックスと乗算を行いMWORK+1番目の領域に格納する。

MWORK ← MWORK+1

MRANK(RANK) ← ON

step 2へ

step 5　GRANK ＝ 0の時

表示データ・リストに論理化因子によりエンティティ・ネームと制御情報をセットする。

GRANK ← RANK

step 2へ

step 6　MWORK ≒ 0ならば，イメージ・データとMWORK番目のワーク領域のマトリックスとを乗算し，さらにそのデータにウインドウイング処理を施して，表示データ・リストにセットする。

MWORK＝0ならばマトリックスの演算は必要としない。

step 7へ

step 7　RANK ← RANK － 1

RANK＝0の時

処理は終了

RANK ≒ 0 の時

NEXTAD(RANK)=0 の時　　step 7 へ

NEXTAD(RANK)≒0 の時

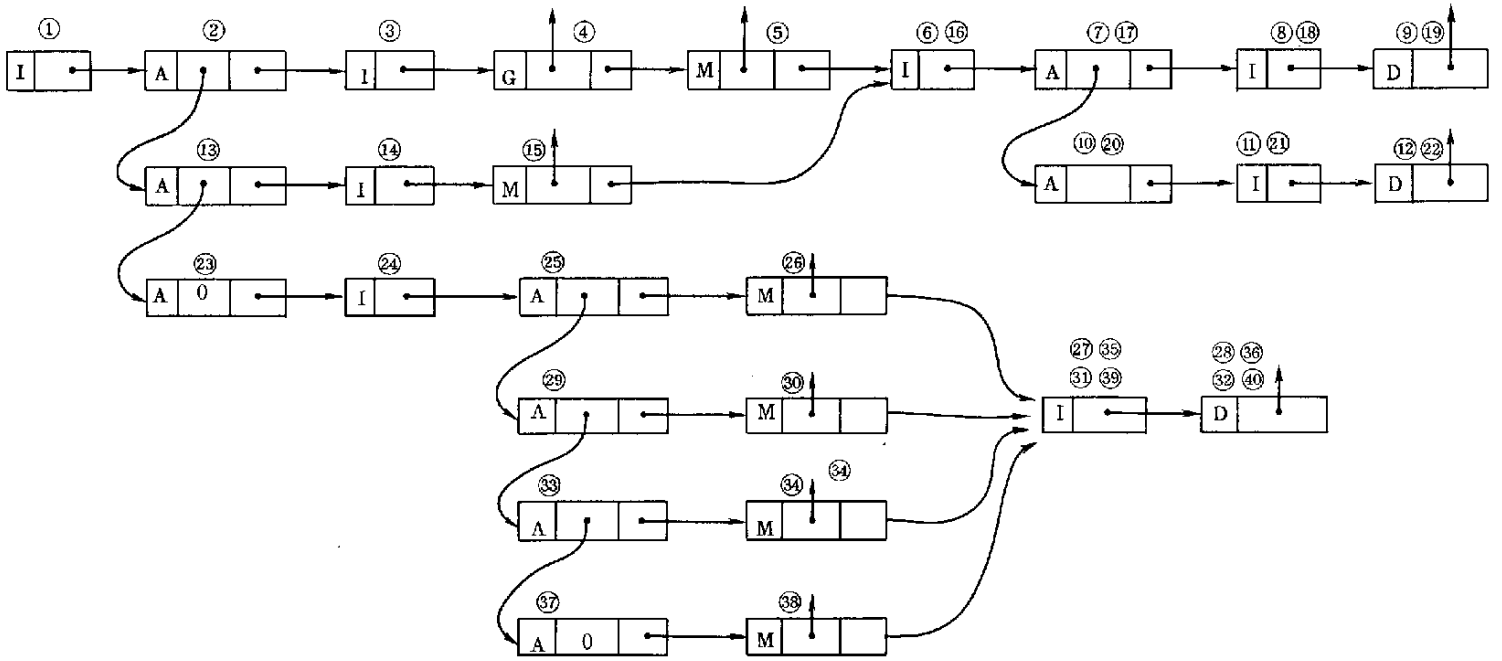
MRANK(RANK) が ONならばMWORK←MWORK−1

GRANK=RANKならばGRANK=0 とし，インターナルなエンティティ・ネームを表示データ・リストにセットする。

step 3 へ

次に〔図３－４５〕に〔図３－２１〕で作られたイメージ構造をトレイスする順序を述べる。①，②，③……の順にトレイスされる。

〔図３－４５〕


①
I
②
A
③
I
④
G
⑤
M
⑥⑯
I
⑦⑰
A
⑧⑱
I
⑨⑲
D
⑬
A
⑭
I
⑮
M
⑩⑳
A
⑪㉑
I
⑫㉒
D
㉓
A
0
㉔
I
㉕
A
㉖
M
㉗㉟
㉛㉙
I
㉘㊱
㉜㊵
D
㉙
A
㉚
M
㉝
A
㉞
M
㉞
㊲
A
0
㊳
M

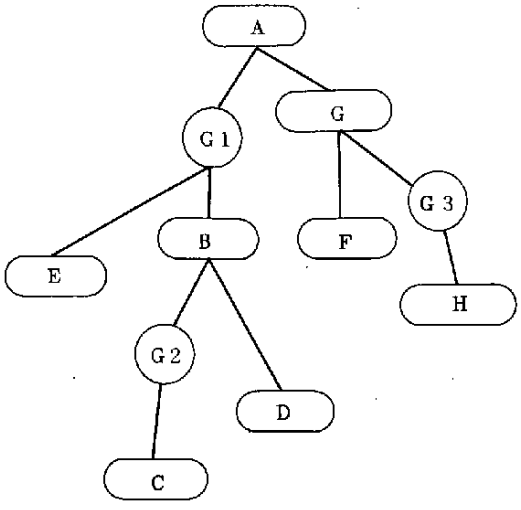
次に論理化因子と変換マトリックスの処理につき述べる。

A．論理化因子の処理

構造をトレイスする場合に論理化因子にのみ着眼して内部処理を考えてみる。論理化因子が階層的につけられた時には，最上層の論理化因子を有効とし，その下に付けられた論理化因子は無視する。

例えば〔図３－４６〕のようにイメージが定義されたとする。

〔図3－４６〕



B<=C|<G2>+D

G<=H|<G3>+F

A<=(B+E)|<G1>+G

イメージＡを出力するステートメントが実行されると表示データ・リストは〔図３－４７〕のようになる。

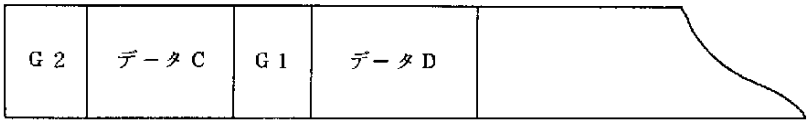
〔図3－４７〕

G1	データE	データC	データD	G I	データF	G3	データH	

ここでＧ１はインターナルにつけられる論理名である。論理化因子がつけられていないイメージに対しては，システムがこのようにインターナルに論理名をつける。

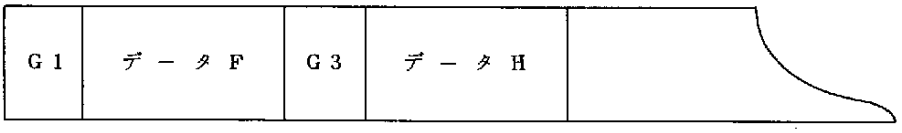
またイメージBを指定すると表示データ・リストは〔図３－４８〕のようになる。

〔図3－４８〕



このようにイメージAを指定した時には，データCはデータEとデータDとともに論理的な一まとまりとみなされたが，イメージBを指定した時には，データCは独自の論理名Ｇ２をもつことになる。同様にイメージCを指定した時には表示データ・リストは〔図３－４９〕のようになり，

〔図3－４９〕



データHは論理名Ｇ３をもつことになる。

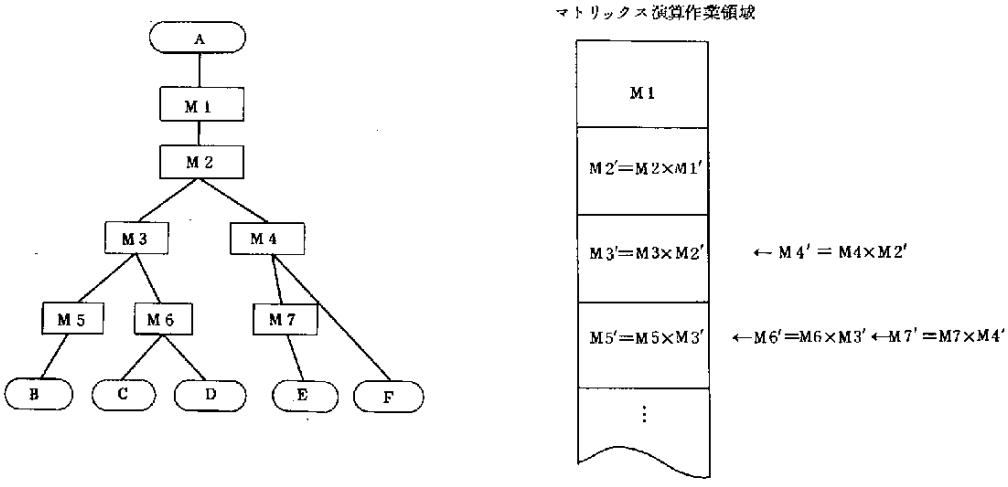
B．マトリックスの処理

構造をトレイスしていく過程で，マトリックスの演算を有効に行うために次のような方法をとっている。

変換マトリックスは構造化されている時は単に変換量を表わすが，トレイスされるとマトリックス演算の作業領域にマトリックスの形で置きかえられる。この作業領域はマトリックスの同じ演算を重複しないように階層毎に演算結果を記憶しておく領域である。演算の過程では現在どの階層にあるかを覚えておき，構造トレイス中にイメージ・データと出会った時，その階層の

作業領域内のマトリックスとかけあわせる。

〔図3－50〕



例えば，〔図3－50〕でイメージAの出力が指定されると，まずM1がマトリックスの形に変換されて作業領域に格納される。次にM_2がマトリックスの形に変換され，M_1とかけあわされ次の領域に格納される。このように逐次マトリックスの演算結果を格納していく。イメージ・データBとぶつかると，このデータは，M_5'とかけあわされ座標変換されたデータとなる。次にイメージ・データC，イメージ・データDのデータを求めるためにM_6とM_3'をかけ，M_5'が格納されていた場所に格納する。このような方法をとることによりマトリックス演算の回数をへらすことができる。

マトリックスの演算は，マトリックスの形が決っているため（例えば0要素の位置は固定），通常の4×4のマトリックス乗算をする必要はない。

3.4.2 ウインドウイング

イメージが定義される座標系は無限に大きなものと考えてよい。しかしディスプレイ画面は限られたものであり，通常1024×1024メッシュの表現

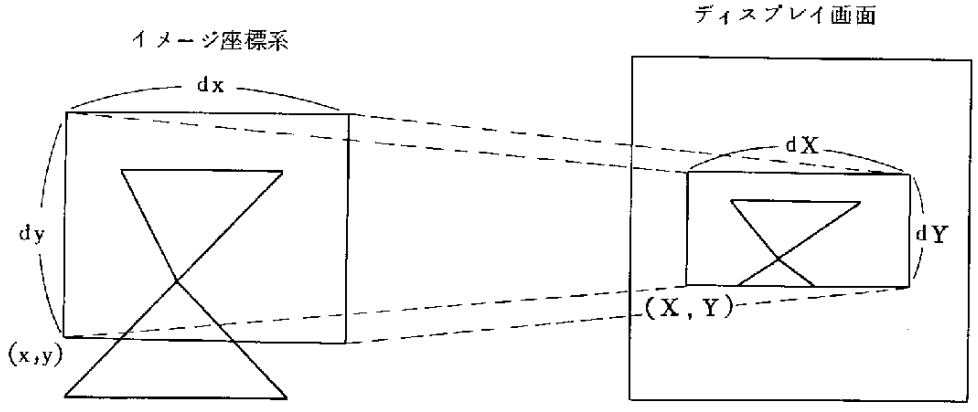
力しか持たない。従ってイメージを表示する際は，イメージ全体をみるために縮小して表示したり，一部分をみるためにその部分を拡大して表示するような手段が必要となる。このように表示したい部分をとりだす操作をウインドウイングと呼ぶ。

UNGLでは「イメージのどの領域をディスプレイ画面のどの領域に表示する」という指定を次のような形式で表現し，イメージ出力文DRAW，ANIMATEに付加することができる。例えば

DRAW imageid.（x，y，dx，dy，X，Y，dX，dY）

アーギュメントは〔図３－５１〕に示すように，イメージ座標系とディスプレイ画面の領域を定義する。領域は一般に矩形とする。

〔図3－５１〕



次にイメージ座標系で指定された領域内のデータをとりだすアルゴリズムと，そのデータがディスプレイ画面の領域内のどの位置にあるかを示す２つの領域間の変換操作を述べる。

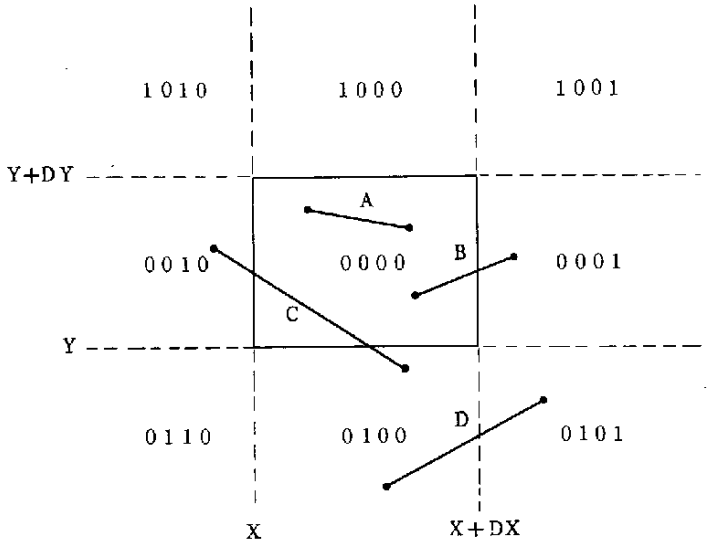
まずはじめに，イメージ座標系における領域内のデータをとりだす方法としては次のような方法を採用した。

A．イメージ座標系のデータを取り出す

第一にイメージを表わしている線分が領域内に完全に入っているか，完全に外にあるか，それとも部分的に内にあるかを判別するために次のようなテストを行う。

線分の両端点が領域に対してどのような位置となるかを示す〔図３－５２〕のようなコードを定める。

〔図3－52〕



線分Ａは両端点のコードが００００，００００で完全に領域内にある。又線分Ｄは完全に領域外にある。このように両端点のコードから次のようなことがいえる。

(1)　両端点のコードが００００の時は完全に領域内にある。

(2)　(1)以外で両端点のコードの論理積が００００でない時は完全に領域外にある。

他の場合には，コードが００００でない点に関して，表示できる領域境界線上の点を次のように求める。

step 1　１０＊＊　の時

ｙ＝Ｙ＋ＤＹ　との交点を求める。

０１＊＊　の時

ｙ＝Ｙ　との交点を求める。

step 2　(1)の交点がＸとＸ＋ＤＸの間に位置すればその点が求める点である。

step 3　step 2 が成り立たなければ

＊＊１０　の時

ｘ＝Ｘとの交点を求める。

＊＊０１　の時

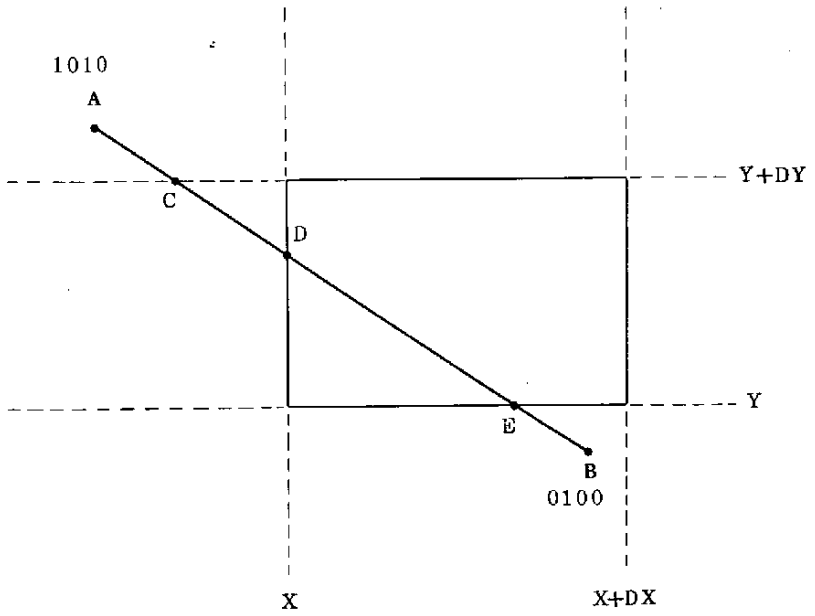
ｘ＝Ｘ＋ＤＸ　との交点を求める。

その点が求まる点である。

このように交点を求める演算は高々２回ですむ。

次に例をあげて説明を加える。

〔図３－５３〕

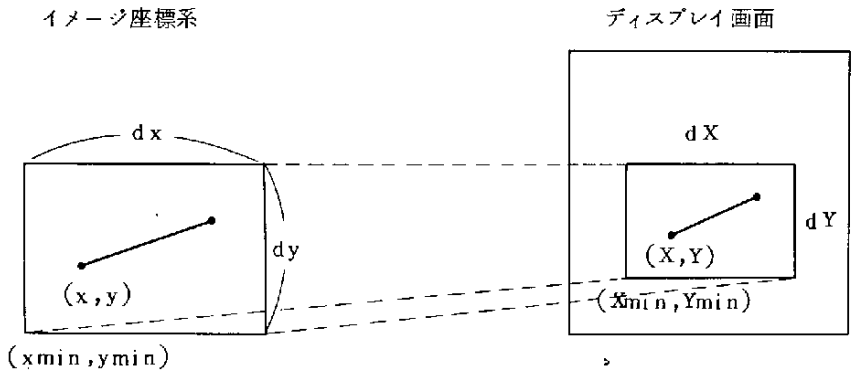


〔図３－５３〕の場合を考えてみると，Ａのコードは１０１０，Ｂのコードは０１００であり両端点のコードは００００でなく，論理積も００００でない。そこで線分ＡＢと直線ｙ＝Ｙ＋ＤＹとの交点Ｃを求める（ｓｔｅｐ １）。点ＣはＸとＸ＋ＤＸの間にないので次にｘ＝Ｘとの交点Ｄを求める（ｓｐｅｐ ２）。同様にＥを求める。その結果線分ＡＢはウインドウイングによりＤＥのみが表示されることになる。

Ｂ．ディスプレイ画面の座標系への変換

このようにして選び出された線分は，〔図３－５４〕に示すようにディスプレイ画面の指定された領域内に位置づけされる。

〔図3－５４〕



イメージ座標系の領域とディスプレイ画面の領域とは必ずしも相似形である必要はない。

イメージ座標系の点（ｘ，ｙ）をディスプレイ画面上の点（Ｘ，Ｙ）に変換する式は次の通りである。

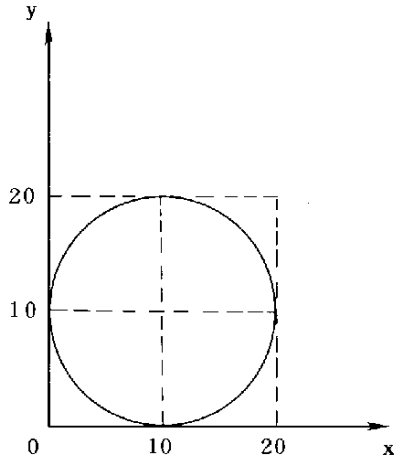
$$X = \frac{dX}{dx}(x - x\,min) + Xmin$$

$$Y = \frac{dY}{dy}(y - y\,min) + Ymin$$

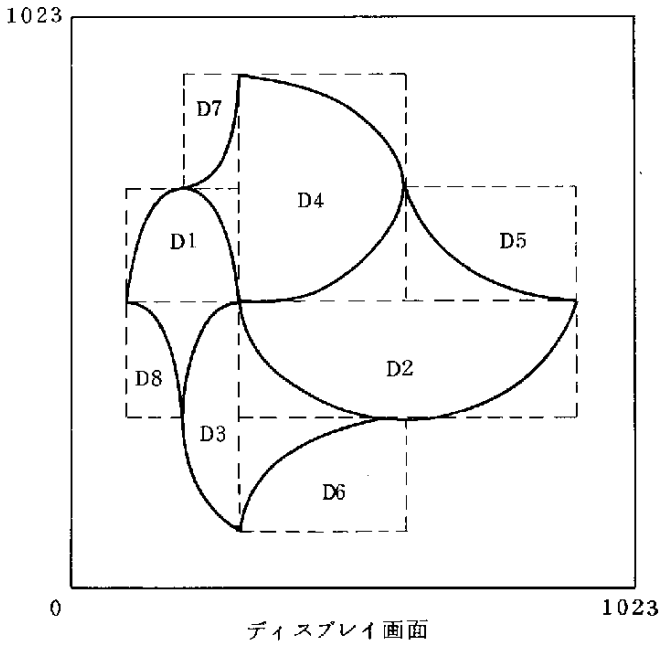
〔図３－５５〕は円をイメージCIRCLEとして定義し，ウインドウイングの機能を利用して複合図形を表示した例で，次のステートメントによって表わされる。

```
DRAW CIRCLE (0,10,20,10,100,500,200,200) ····· D1
DRAW CIRCLE (0,0,20,10,300,300,600,200)  ····· D2
DRAW CIRCLE (0,0,10,20,200,100,100,400)  ····· D3
DRAW CIRCLE (10,0,10,20,300,500,300,400) ····· D4
DRAW CIRCLE (0,0,10,10,600,500,300,200)  ····· D5
DRAW CIRCLE (0,10,10,10,300,100,300,200) ····· D6
DRAW CIRCLE (10,0,10,10,200,700,100,200) ····· D7
DRAW CIRCLE (10,10,10,10,100,300,100,200)····· D8
```

〔図３－５５〕



イメージ座標



ディスプレイ画面

3.4.3 アニメーション

ANIMATEステートメントは次のようにムーヴィング情報とウインドウイング情報をもつ。

ANIMATE IMAGE(ムーヴィング情報),(ウインドウイング情報)

そして，イメージをディスプレイ画面上でリアル・タイムに連続的な平行移動(ムーヴィング)をさせることができる。ムーヴィングはANIMATEステートメントが実行と同時に開始され，ムーヴィングが完了するとプログラムに終了割り込みの形で通知されるようになっている。

A．ムーヴィング情報

ムーヴィング情報の形式は，少ない情報量で複雑な動きが表現できることが望まれるが，UNGLでは次の形式をとっている。

ANIMATE image id.(i_1,j_1,k_1,l_1,XAR_1,YAR_1〔,
i_n,j_n,k_n,l_n〔,XAR_n,YAR_n〕・・・)

ここで各アーギュメントは次のような意味をもつ。

i ： ループのレベルを与える。0～9の範囲で記す。0は最小レベルでループの入れ子の一番内側となる。

j ： ループ回数を与える。1～∞(∞の時は0を与える)

k ： ループ時の△x，△y符号反転，及び△x，△yの入れ替え指定

0 変化せず
1 △y符号反転
2 △x符号反転
3 △x，△yの入れ替え
4 1と2の組合わせ
5 1と3の組合わせ
6 2と3の組合わせ
7 1，2，4の組合わせ

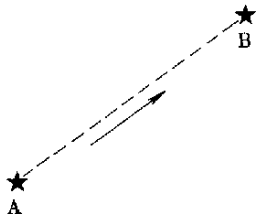
l ： このループに含まれる△x，△yの組数

XAR ： 1で指定した数の△xの要素を含む配列名または変数名で，この値はディスプレイ画面上の実際の移動量を示す。

YAR ： XARの△xが△yに変ったもの。

例えば〔図3－56〕のようなムーヴィングを行う時，ANIMATE STAR（0，50，0，1，4，3）と表わされる。△x，△yはそれぞれ4，3メッシュで50回のムーヴィングを行うことを意味している。これは単にイメージSTARがAからBへ移動する簡単な例である。

〔図3－56〕

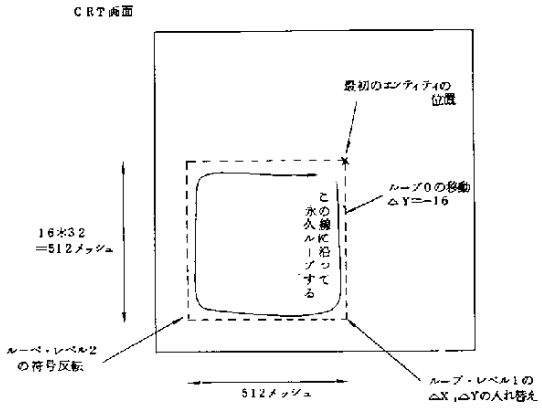


次に少し複雑なムーヴィングの例を示す。

ANIMATE STAR（0，32，0，1，0，－16，1，2，4，0，2，0，3,0）

上記のようなムーヴィング情報によってエンティティは〔図3－57〕のような動きをする。つまり0レベルのループはy方向のきざみが－16で32回繰り返し，1レベルのループはループ時にx方向のきざみとy方向のきざみを入れ替え，2回繰り返すことを意味している。さらに2レベルのループはループ時にx方向のきざみとy方向のきざみの方向反転を行い無限に繰り返す（回数0）ことを意味している。

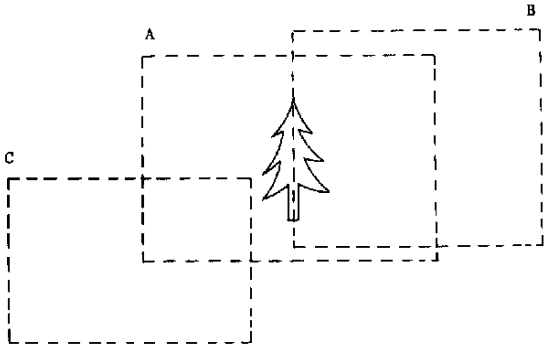
〔図3－57〕

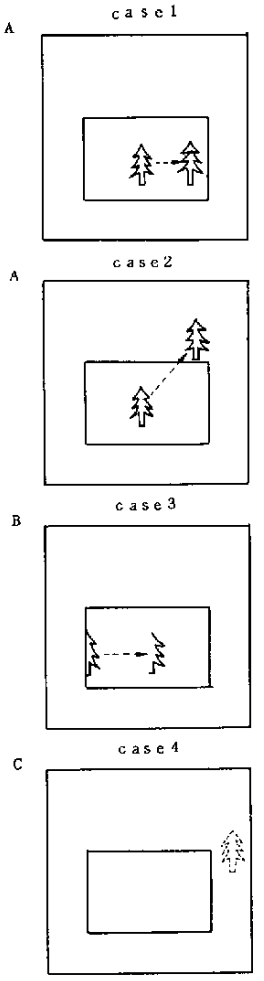


B. ANIMATEにおけるウインドウイング処理

ANIMATEステートメントの実行に於て，エンティティがディスプレイ画面上を動き座標が序々に変更されるために，途中からウインドウイングの領域の外に図形がはみ出したりあるいはその逆に領域の外から入ってくることも考えられる。〔図３－５６〕に示すように，case １の場合はムーヴィング後もエンティティはディスプレイ画面の領域内に納まるために問題はない。case ２の場合には領域の境界でエンティティは消えずにエンティティは領域の外に出てしまう。case ３の場合には，ムーヴィングの始点でウインドウイング処理によってイメージの一部分が切り取られ，それがエンティティとしてムーヴィングされてしまう。case ４の場合はムーヴィングの始点においては領域外にあり，ムーヴィング後には，領域内に存在すると考えられる場合である。case ２ ～ case ４の場合には，ムーヴィングのきざみ毎にウインドウイング処理が必要となり，リアル・タイムで動くムーヴィングの意味がなくなる。もしそのような動きを与えたいならば，ウインドウイング情報にきざみを与えて，変化させDRAWステートメントをループさせて表示することにより，多少速度はおちるがエンティティの動きを表現することはできる。よってANIMATE ステートメントの処理においてはムーヴィング開始時にウインドウイング処理をし，そのエンティティが動くような方法をとっている。ウインドウイング情報の指示方法は 3.4.2 を参照のこと。

〔図3－58〕





C．ムーヴィングの完了

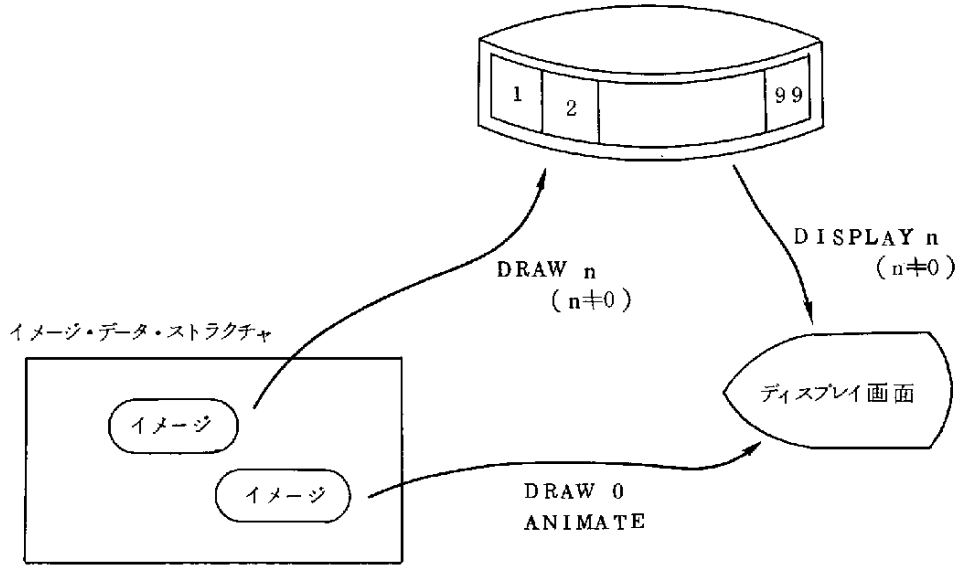
ムーヴィングが終了すると，ムーヴィング完了割り込みがかかる。従ってANIMATEを使用する場合はANIMATEステートメントのすぐ後でAWAITステートメントをだすことが望ましい。AWAITステートメントでwait状態とすることにより，割り込みの同期がとられる。そのためにはムーヴィング完了割り込み要因MVEDはONステートメントで登録されていなければならない。割り込みがかかると登録されている割り込み処理ルーチンに制御はわたされる。もしAWAITがだされていないと，ANIMATEの次のステートメントが実行され，ムーヴィングが完了すると非同期に割り込みが生じる。AWAITステートメントがない場合は，割り込み処理ルーチンはリエントラントに組まれていなければならない。

3.5 フレーム管理

3.5.1 フレームの概念

イメージが出力される媒体をフレームと呼ぶ。フレームは０～９９（整数）で表わされる識別名をもち，ディスプレイ画面一画面に相当する表示データが格納される。特にフレーム０とは，ディスプレイ画面そのものであり，フレーム０に出力するということはディスプレイ画面に表示することである。ディスプレイ画面に表示されたイメージはプログラム終了と同時に消滅してしまう。一方他のフレーム１～９９はディスク上に記憶されているもので，フレームへの出力はディスクへの出力を意味し，その時点ではディスプレイ画面には表示されない。そのフレームを見たい時には，フレーム出力ステートメントDISPLAYを実行することにより，ディスプレイ画面に表示することができる。このことからプログラム実行中にも同じ画面を繰り返し表示することもできるし，必要な画面はフレームｎとして記憶しておき，後日ディスプレイ画面に表示することも可能である。〔図３－５９〕

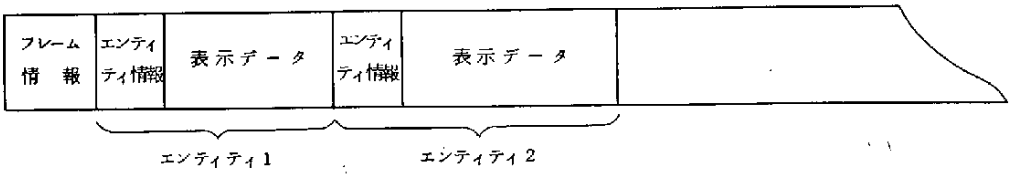
〔図3－５９〕



ここで1つ注意すべきことは，DISPLAY nによってフレームnのエンティティがディスプレイ画面に表示されるが，表示されたエンティティはフレーム0に属することになる。つまりフレームn内のすべてのエンティティをフレーム0に付け加えるとも考えられる。この場合もディスク上にはフレームnの元のイメージ情報は残されている。

フレームは表示データ・リストと同じ型式をとる。

〔図3－60〕



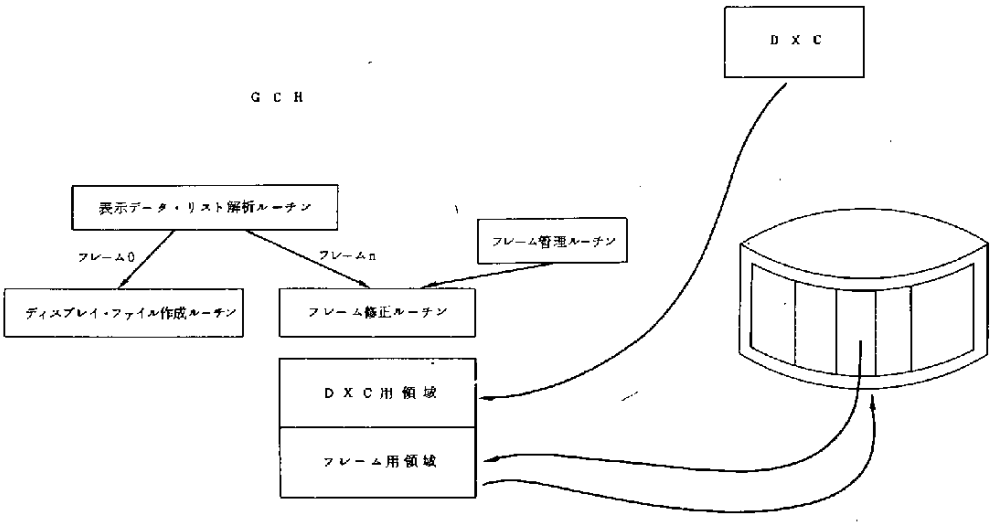
このようにフレームに出力されたイメージは，イメージ単位はくずされ，エンティティ単位にまとめられ，この段階では，イメージ構造も当然失われている。従ってこのフレーム内の図形のアクセスもエンティティ単位で行われ，論理化因子で指定した論理名が識別名とされる。

3.5.2 フレーム管理

フレーム内の図形表示用制御情報の変更，図形の消去はイメージの構造とは関係なく，フレーム内の情報を修正するだけで実行できる。そのことからフレーム修正及びフレーム管理はGCHが行っている。DXC経由で転送されてくる表示データ・リストは，フレーム0への出力の場合は，直接ディスプレイ・ファイル作成ルーチンに渡されるが，他の場合はフレーム修正ルーチンに渡される。DXC用領域にはDXC経由で転送されてきたデータが読み込まれる。フレーム用領域には，指定されたフレーム番号のデータが読み込まれ，二つの領域内のデータを比較修正する。ただし，修正するフレームが既にフレーム用領域に読み込まれている時は，改めて読み込む必要はない。こ

のようなフレームの管理，ディスクへの読み書きはフレーム管理ルーチンが行っている。１フレームはディスク上では１トラックを占め，フレーム番号によってディスク上のアドレスが決定できるので，フレーム番号と記憶アドレスとの対応表は必要としない。これらの関連を〔図３－６１〕に示す。

〔図3－６１〕



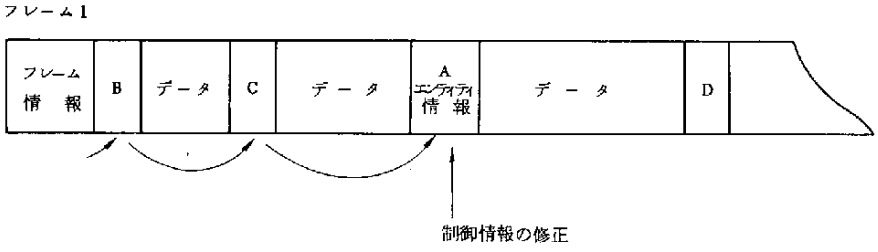
3.5.3 フレームの修正

A．制御情報の変更

制御情報の変更はCONTROLステートメントによって行われる。指定されたフレーム内に目的のエンティティが無ければそのステートメントは無視される。次に簡単に内部処理について述べる。

〔図3－62〕

CONTROL　1，<A，－3>



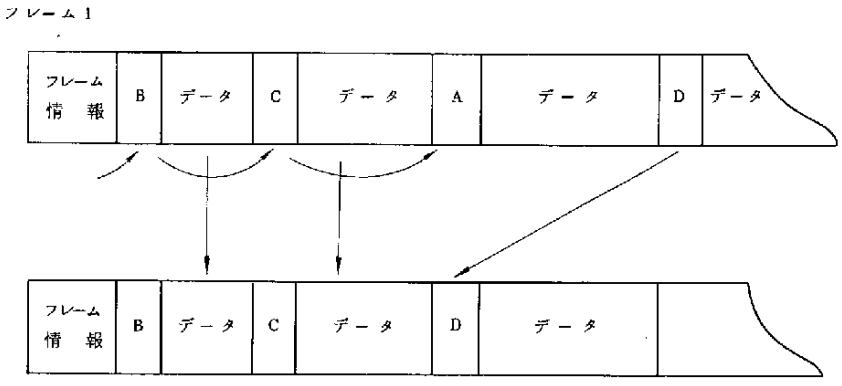
各エンティティはエンティティ情報としてエンティティ長をもっているので，〔図3－62〕に示すようにフレームの先頭から順次エンティティを検索し，制御情報を修正する。CONTROLステートメントによる修正は表示時の制御情報の修正であって，対応するイメージ構造内の論理化因子の制御情報は修正されない。

B．エンティティの消去

エンティティの消去はDELETEステートメントによって行われる。エンティティが消去されるということはディスプレイ・ファイル内から対応するエンティティの部分を取り除くということである。〔図3－63〕に簡単に内部処理を示す。

〔図3－63〕

DELETE 1,<A,－3>

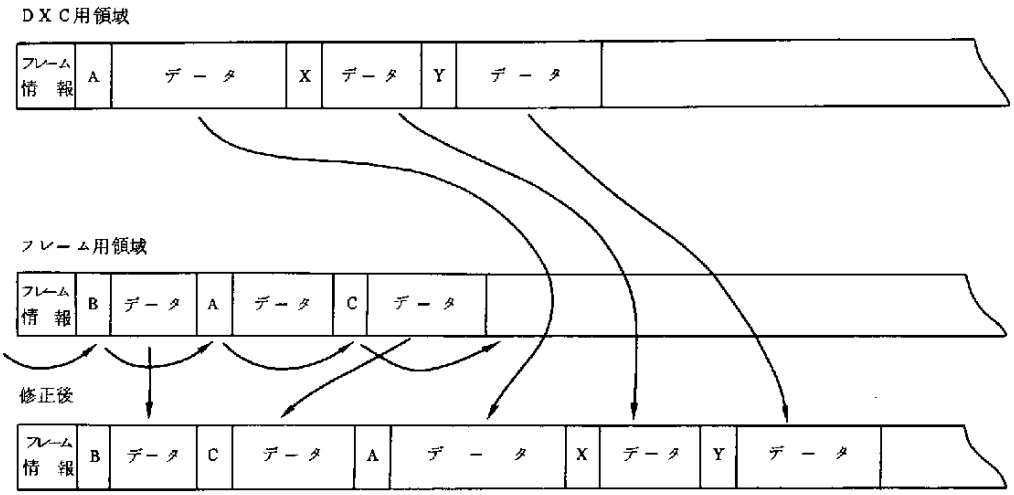


エンティティAが取り除かれ，後につづくエンティティが前につめられる。

C．エンティティの修正

DRAWステートメントにより表示データ・リストがDXC経由で転送され，DXC用領域に読み込まれる。フレーム用領内の指定されたフレームのデータと比較し修正が行われる。既に存在するエンティティは追加する。〔図3－64〕に簡単な例をあげる。　DRAW 1, A, X, Y

〔図3－64〕



ここでAは既に存在するエンティティであるから，エンティティの情報とデータを書きかえる。修正するエンティティの長さは既存のエンティティの長さとは異なる場合が多い。従って変更の対象となるエンティティは取り除き，次のエンティティを前にもってくる。変更エンティティを全て取り除いた後に，修正のエンティティ全てを付加する。それによって新しく付加されるエンティティと修正のエンティティが整う。

3.6 ロード・シェアリング

UNGL は H−8400をホスト・コンピュータ，H−8811をサテライト・コンピュータとするマルチ・プロセッサのもとで動作することは前にも述べた。そのために各プロセッサの処理分担（ロード・シェアリング）をどのように決めるかが問題となる。ここでは図形処理に観点をおいて述べてみよう。

通常図形処理（イメージ・データ・ストラクチャも含む）はすべてサテライト側にもつべきだということがよくいわれる。しかしマトリックスの演算，イメージ・データ・ストラクチャの領域などを考慮に入れると，サテライト側のコアサイズ，計算処理能力では実現は困難である。そこでサテライトの処理能力との兼ね合いで，図形処理の中のどの程度の処理を受けもたせるかが問題となる。例えばイメージ・データ・ストラクチャをホスト側に持たせ，ストラクチャとは独立に動くことができる処理をサテライト側に割りあてるとする。すると表示データ・リストはイメージ単位がくずされエンティティ単位で構成されているという性質を利用して次の処理をサテライト側にもってくることができる。

① 表示データ・リストからディスプレイ・ファイルを作成する。

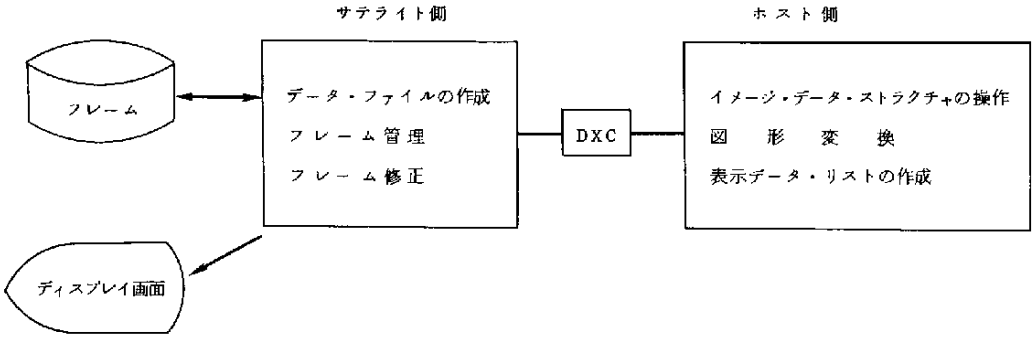
② フレーム管理

③ フレーム修正

従ってフレームはサテライト側のディスクに記憶される。

UNGLではこのようにイメージ・データ・ストラクチャをホスト側のコア内に置くことから出発し，〔図3−65〕で示すようなロード・シェアリングを行っている。

〔図3－65〕


サテライト側
ホスト側
フレーム
データ・ファイルの作成
フレーム管理
フレーム修正
DXC
イメージ・データ・ストラクチャの操作
図形変換
表示データ・リストの作成
ディスプレイ画面

4. 割込み処理

4. 割込み処理

インタラクティブなシステムと言うのは，ある種の処理結果を見て，ユーザが新しい処理を要求しそれを実行する。更に次の要求を出す……と言う具合に，新しい要求に対してコンピュータが即座に応じる事が望まれる。特にグラフィック・ディスプレイを使用したインタラクティブ・システムは，可視的であり且つグラフィカルなものであるので，次の事が強く要求される。

1. 装置の特性を生かす

   CRT，インタラクティブI／Oなどが持つ特性が十分に活用される事。

2. 汎用性がある

   システムがマシン・インディペンデントであり，しかも各種のアプリケーションに共通に利用できる概念であることが望ましい。

3. 拡張が容易である

   急速に進歩するオペレーティング・システムや，ハードウエアに対処できるものとするためには，拡張可能なシステムであるという要求が強くなる。

4. 使用効率が高い

   グラフィック・システムを使用する事によって，そのコンピュータでマルチで行なっている他のバッチジョブなどに大きな影響を与えない事が必要である。その反面，グラフィック・システムもリアル・タイム・システムとして常に使用できる状態にある事が望ましい。

5. 速やかなレスポンス

   ユーザが，グラフィックCRT画面の前にすわってシステムとのインタラクティブな処理を行なおうとする場合，レスポンス時間が人間工学的に許容されうる時間内にある事。

インタラクティブ・システムはこれらの要求を考慮に入れて設計がなされなければならない。UNGLにおいては，上記の事項を充分考慮に入れ，特に割込み処理についてはそれらの要求を可能な限り取り入れている。

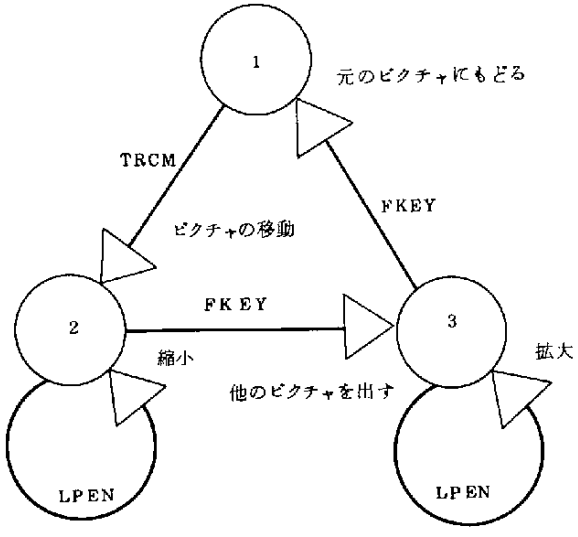
4.1 割込みモデルの概念

インタラクティブ・プロセスについてW．M．Newmanは次の様に述べている。「どんなインタラクティブ・プロセスの記述でも，ある入力に対するシステムのレスポンスとして定義される。このことからインタラクティブ処理用言語をアクションとリアクションによって述べる事は非常に有効である。」アクションとは，レスポンスを生じるような1つの入力であり，リアクションとは，このアクションに対する反応として規定され，システムの状態によって同じアクションでも異なるリアクションを引き起こす。この事からシステムを有限オートマトンとして扱い，リアクションがプログラムの状態（program state）によって規定されると考えれば，インタラクティブ・プロセスの表現が明確にされる。

有限オートマトンをステート・ダイヤグラムによって示し，割込みモデルの概念を〔図4－1〕によって明確化してみる。

縮小，拡大される図形を，ライトペン，ファンクションキー，トラッキングマークによって次の順序でつくる

〔図4－1〕



(1) ピクチャを表示する。

(2) トラッキング・マークを任意の位置に移動し，ピクチャをそこまで移動する。

(3) ピクチャをライトペンで指示するたびピクチャが縮小する。

(4) ファンクション・キーによって他のピクチャを出す

(5) ピクチャをライトペンで指示する度にピクチャが拡大する

(6) ファンクション・キーを押す事によってもとのピクチャにもどる。

この例で，プログラム状態１では，(2)のアクション，リアクションが許され，状態２では(3)(4)，状態３では(5)(6)のアクション，リアクションがそれぞれ許される。この事は同じライトペンによるアクションであっても，状態２では縮小，状態３では拡大を指示することを表わし，同じアクションで異なったリアクションを引き起す事を示している。〔図４－１〕の①②③をプログラム状態とすると，割り込み処理は，プログラム状態の遷移と，その中で許されるアクション，リアクションという形で表現する事ができる。この割込みモデルの概念によって，インタラクティブな処理過程を明確化し，ユーザの割込み処理プログラムの遷移を簡単に把握することができる。

すなわち，ステート表現は，インタラクティブなプログラムを記述する方法として便利であり，割込みモデルを一般に扱われる多種のモデルの１形態として表現する事ができる。

UNGLの割込みモデルは，ユーザの割込み処理プログラムの処理過程を明確にし，プログラム作成上の困難な点を除いている。以下に，インタラクティブ処理における割込み処理の概要と，割込み処理上の問題点を述べる。

4.2 割込み処理の方法

割込み処理に於ては，その処理形式と制御方法の２面が考えられる。

4.2.1 割込み処理形式

割込み処理形式には，非同期処理と同期処理の２つの形式があり，割込みが発生した時のシステムの動作の性格を表わす。

a．非同期処理

割込みが入った時点で直ちにその処理に移るような処理形式例えばプログラムの処理中にループに入ってしまったとか，処理時間が長すぎ，予期した通りの画面表示が行なわれない場合などには，この非同期処理系を採用しなければならない。しかしながら非同期処理においては，プログラム・リソースにリエントラント性を要求されるので，既存の言語をベースとするグラフィック言語では，実現が困難となっている。又非同期処理が可能かどうかは，ベースとなるグラフィック・オペレーティング・システムの性格によって定まってくる場合もある。

b．同期処理

割込みが入った時点で，直接ユーザの割込み処理プログラムに割込まず，システムがこれをキューイングしておいて，必要な時点で（同期をとって）ユーザ・プログラムに制御を渡す方法と，プログラムが割込み待ちの状態にいる時のみ割込み処理プログラムに制御を渡す方法と２種類の処理形式がある。非同期処理に比較すれば即応性には欠けるが，ソフトウェア的にかなりの部分をカバーし得ると同時に，汎用の観点からは，より多くのオペレーティング・システムに受け入れ可能な形式である。

4.2.2 割込み制御の方法

割込みが発生した時の割込み制御は一般には，システムが持つ割込み制御情報によってなされる。この割込み制御情報は具体的に，各種の割込み要因に対して，その割込みを許容するか禁止するかの情報と，割込まれた要因と，割込み処理プログラムとの対応関係を指示する情報を含んでいる。この割込

み制御は，一般に割込み処理過程の制御にもつながる。

割込みの処理は，割込み制御情報の登録，同期処理の場合キューイングされた割込みの取り出し，および割込み待ちの状態をどのような形式で表現するかという問題に集約される。この扱いは表現上次の3つに分類される。

a．割込み制御表を持たないもの

ユーザ・プログラムからシステムに与える指示は，割込みの禁止又は許容だけであり，システムはこれに従って，割込み要因のキューを作成しておく。即ち，割込みが発生しても，直接ユーザの割込み処理プログラムに割込みをかけず，ユーザ・プログラムの指示に従って，キューから割込み情報を取り出す事ができるようになっている。割込み情報は，ユーザ・プログラムによって解析され，それに応じた処理がとられる。

b．割込み制御表を明示的に作成するもの

割込み要因と割込み処理プログラムとの対応をシステムに登録するための指令があり，ユーザはこれによって割込み制御表を作成する。又割込みの禁止，許容などを指示する命令を持つのが一般的である。この場合，割込みが入った時点で，直接ユーザの割込み処理プログラムに制御を渡す方法をとれば非同期処理となり，プログラムを割込み待ちにしておき割込みが入ってくるのを待っている方法をとれば同期処理となる。

c．割込み制御表を意識させずに，これをプログラム・ステートという概念で扱うもの。

インタラクティブ・システムを有限オートマトンと考え，アクション・リアクションをそれぞれオートマトンへの入力，出力として扱い，プログラム・ステートを，アクション，リアクションで定義したものである。ユーザは割込み制御表を意識することなく，ただプログラム・ステートを定義すればよい。これに対しシステムは自動的に割込み制御表を作成し，それによって割込みの制御をおこなう。

4.3 UNGLの割込み処理機能

インタラクティブ・システムを効果的に実現するために，UNGLでは，割込み処理として次の様な機能をもつ。

(1) スタティックなプログラム・ステートの定義

割込み要因とユーザの割込み処理プログラムの対応付けを定義する。

(2) スタティックなプログラム・ステートの遷移

プログラム・ステートの遷移の状態を，プログラム・ステートの中でスタティックに定義する。

(3) ダイナミックなプログラム・ステートの遷移

プログラム・ステートの遷移をダイナミックに行なう。

(4) プログラム・ステートの変更

既に定義されたスタティックなプログラム・ステートをダイナミックに変更する。

(5) 割込み情報の詳細を調べる

割込み要因の種類及び内容，例えばライトペンのポジションとか，どの図形がピックされたかなどを調べる。

(6) 非同期処理

非同期に割込まれるアクションに対してのシステムの応答及び，必要ならばユーザの割込み処理プログラムへの制御の受渡しを行なう。

(7) 割込み抑制のコントロール

プログラム・ステートで定義された割込み要因に対して，割込みの抑制，解除などを行なう。

(8) CGOSとの間のデータ転送

ユーザ・プログラムと，GJMON(Graphic Job Monitor)との間のデータ転送及び，GJMONから受け取ったデータの解析と，数値データの変換などを行なう。又，GJMONから依頼されてグラフィック・プログラムの制御の処理も行なう。

以上がUNGLの割込み処理機能であるが，これらの機能の処理プログラムはFORTRANコンパイルの終了後，リンケージ・エディタによってユーザ・プログラムにリンケージ・バインドされる。

以下にUNGLの割込み処理形式，割込み処理過程について述べる。

4.3.1 UNGLの割込み処理形式

インタラクティブなプログラムはオペレータの指令による，非同期な割込みに対して柔軟に対処できなければならない。すなわち，完全な意味でのインタラクティブ・モードとは何らかの外部割込みに対して，プログラムの処理中であっても，一担中断し，要求のあった処理を先に済ませてからもとの処理を再開し，その間に混乱が生じない事が必要であり，いわゆる会話型のRead and Replyとは処理を異にする。この場合には制御が渡る割込み処理プログラムはリエントラント構造でなければならない。この事を考慮に入れた上でUNGLでは割込み処理形式は非同期処理と同期処理の両者を使用できるように作成されている。

a．非同期処理

プログラムがディザーブル（disable）状態で実行されており何らかの割込みが非同期に発生された場合に，ユーザの割込み処理プログラムがリエントラントである場合に限り，そのプログラムへ制御を渡す。もしそうでなければ，その割込みは無視される形となる。UNGLユーザは，FORTRAN言語の使用が一般的であると考えられるが，非同期の割込み処理プログラムをユーザが作成すれば（例えばアセンブラ言語で）非同期処理も可能となる。

b．同期処理

同期処理は，①割込みが発生する迄，プログラムを待ち状態に置き，割込みが発生した時点で，その割込要因を解析し，指定の処理プログラムへの制御を渡す方法と，②プログラムを割込み待ちの状態にしておき，割込みが発生するとその割込み要因をしらべ，指定されたものであるか否かを判断して，ユーザ・プログラムに制御をもどす方法との2種類があり，全て同期をとっ

た処理形式となっている。一般のUNGLユーザは，この同期をとった処理形式だけで十分インタラクティブ・システムを実現する事が可能である。ユーザのプログラムの実行中の状態は，割込み待ちの状態と，割込み処理プログラム実行中の状態及び割込み処理プログラム以外のプログラムの実行中の状態の3つに分けられる。このうち割込み待ち以外の状態で生起する割込みに関しては，非同期処理のみが認められる。又，割込み待ちの状態では，システムの規定する割込みループを実行している状態であり，同期処理は全てこの形式で処理されている。又割込み処理プログラム実行中のモードでは，次の同期をとった割込みを受けつけようとする場合には，プログラムが復帰するか，もしくは待ち状態を再度指定するかしなければならない。

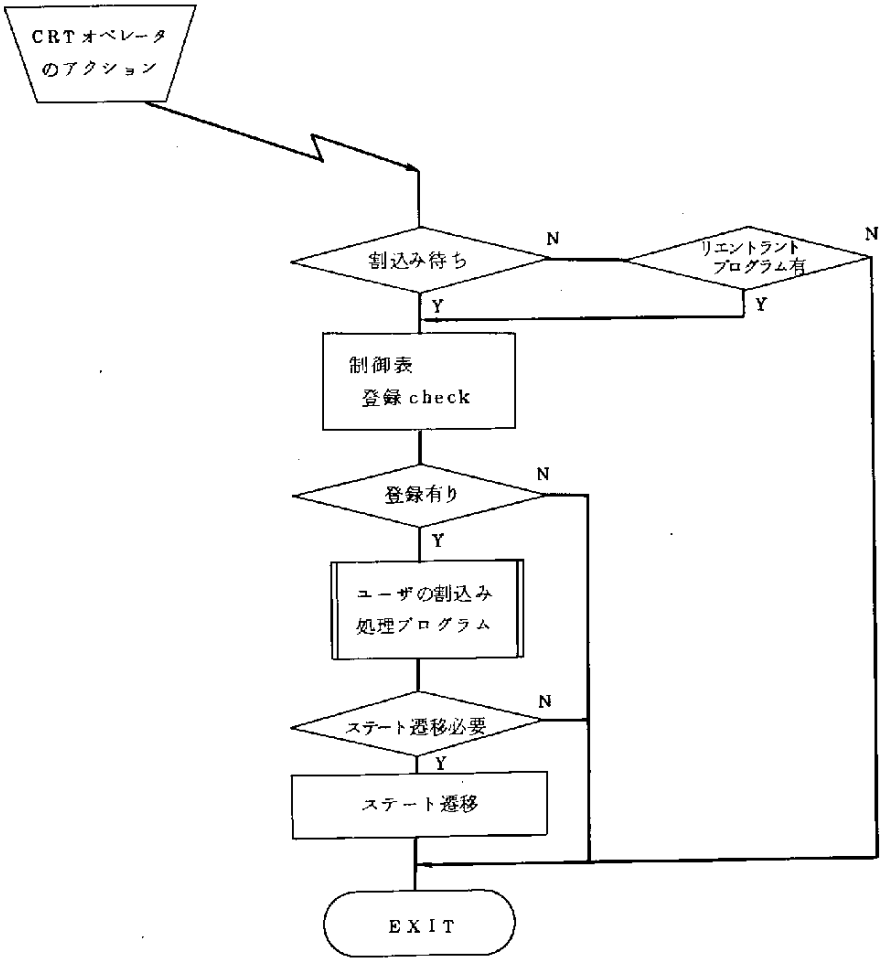
以下に具体的なUNGLの割込み処理手順について述べる。

4.3.2 UNGLの割込処理手順

UNGLの割込み処理機能モジュールは，インタラクティブ機能を果すために，GJMONとの間でプログラム間のデータ転送を行なっている。すなわち，グラフィックCRTのオペレータが発した何らかのアクションは，GCHからDXCを経由してGJMONに知らされ，GJMONからユーザのアプリケーション・プログラムにリンケージ・バインドされたUNGLの割込み処理モジュールに非同期割込みで知らされる。つまりCRTのオペレータが発した何らかのアクションは，ユーザ・プログラムがどの様な処理中であっても，待たされる事なく直ちにUNGLの割込み処理モジュールに渡される。割込み処理モジュールは，プログラムが割込み待ちの状態であれば，アクションが出された割込み要因を割込み制御表から探す。ここに登録されている割込み要因とアクションが出された割込み要因が一致したら，割込み情報をメッセージ・受取りバッファに転送し，ユーザの割込み処理プログラムにコントロールを渡す。このようにして制御を渡されたユーザの処理プログラムは，受け取った割込み情報に従って必要な処理を行ない，UNGLの割込み処理モジュールに返ってくる。ステートの遷移が必要な場合はその後でステート遷移

を行なう。このステート遷移は具体的には，指定されたステートを割込み制御表に登録する事を指す。このアルゴリズムを〔図４－２〕に示す。

〔図４－２〕 割込み処理手順



4.3.3 UNGLの割込み処理過程

UNGLでは割込み処理過程を次の２つの形式で表現する。

a．プログラム・ステートを意識しない方法

ONステートメントによってシステムに割込み処理に関する動作を与える

もので，ＯＮステートメントの割込み制御に関する情報を直接システムの持つ割込制御表に登録する。これは簡単な割込み処理に有利である。

b．プログラム・ステートを意識する方法

割込み処理過程をプログラム・ステートの遷移という事で表現する方法で，あらかじめプログラム・ステートを定義し，システムの作成する割込み制御表は意識させず，プログラム・ステート表によって割込みを取扱う。この表現によると複雑な割込み処理過程をマクロに把握する事ができる。

〔図４－３〕及び〔図４－４〕にステート・テーブル，割込み制御表を示す。

〔図4－３〕 STATE テーブル

S.No	AT.No	FKEY・F	LPEN・F
DIAL・F	TRCM・F	MVED・F	MESG・F
AS・No	FKEY・R		
〃	LPEN・R		
〃	DIAL・R		
〃	TRCM・R		
〃	MVED・R		
〃	MESG・R		

S・No ：STATE No.（1～99）

AT・No：ATTENTION CODE No.（1～6）

FKEY・F：ATTENTION FLAG
MESG・F：DCL. OR NOT AND LOCK OR NOT
DCL='FF', LOCK='FO',
INIT='OO'

AS・No：AFTER STATE No.（1～99）'FF'=non Def.

FKEY・R：WHEN ATTENTION IS DONE
MESG・R：ROUTINE POINT

〔図4－4〕 割込み制御表

AT・C	AT・R
MSK	MSG・A

AT・C：ATTENTION CODE

MSK ：MASK FLAG

AT・R：ATTENTION ROUTINE

MSG・A：MASSAGE BUFFER

4.4 プログラム・ステート

割込み制御表の内容によって定義づけられるプログラム状態をその時のプログラム・ステートといい，ユーザのアクション，リアクションはプログラム状態によって処理される。又，割込み処理過程の遷移は，プログラム・ステートの遷移の制御をうける。プログラム・ステート記述による割込みモデルの作成上の利点及び欠点を述べる。

a．利　　点

(1) プログラムの状態遷移の把握が容易

プログラムが現在どの状態にいるかをアクション，リアクションによって容易に認識できる。

(2) ドキュメント性がある

プログラミングされたものによって，簡単にステート・ダイヤグラム表現をする事ができる。

(3) 記述の容易さ

複雑な過程を記述する場合にも，スタティックな定義によって比較的簡単に表現可能であり，その時許されるアクション，リアクションが明確になる。

b．欠　　点

(1) 共通データの受け渡しが面倒

割込み処理過程がプログラム単位として独立しているので，共通データの受け渡しが面倒となる。

(2) 単純な割込み処理過程に向かない。

単純な割込み処理過程では，プログラム状態の遷移という概念まで必要とせず，単にアクション，リアクションの関係だけで処理過程を表現すればよい。

プログラム・ステートはユーザによって定義され，プログラムはその遷移のコントロールを受ける。プログラム・ステートの基本的な過程は〔図4－5〕に示されるように，各々の状態の間に起こる許されるアクションと，そ

れによって引き起されるリアクションの状態の変化を示している。ここでプログラム・ステートの定義と，プログラム・ステートの遷移について具体的な例をあげて説明する。

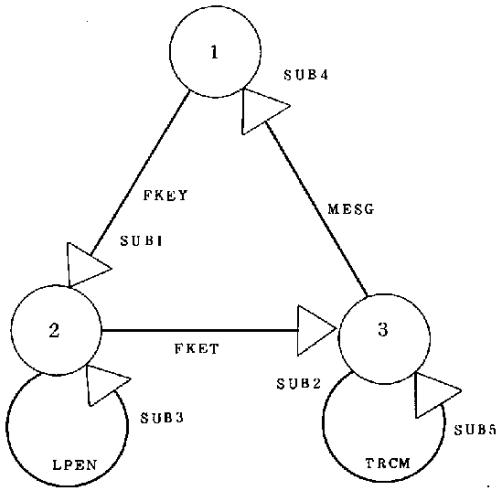
4.4.1 プログラム・ステートの定義

プログラム・ステートは，ユーザによって先ずスタティックに定義されるが，その定義されたプログラム・ステートのダイナミックな変更も許される。プログラム・ステートの定義は，DEFINES文からENDS文の間で行われ，ステート毎に割込み要因と処理プログラムの関係及び，割込み要因とステート遷移の情報が保持される。又，割込み要因は次の6種のキーワードによって類別される。

FKEY	ファンクションキー割込み
LPEN	ライトペン割込み
DIAL	ダイアル割込み
TRCM	トラッキング・クロス割込み
MVED	アニメーション終了割込み
MESG	メッセージ割込み

次に具体的なプログラム例を〔図4－5〕に示し，プログラム・ステートの定義とステート遷移について示す。

〔図4－5〕 スタティックなプログラム例




```
(1)DEFINES
(2)STATE 1
(3)   ON FKEY DO SUB1, 2
    STATE 2
      ON FKEY DO SUB2, 3
      ON LPEN DO SUB3
    STATE 3
      ON MESG DO SUB4, 1
      ON TRCM DO SUB5
(4)ENDS
```

＝プログラム説明＝

(1) プログラム・ステートの定義の開始を示す

(2) プログラム・ステートを定義する。

(3) 割込み要因とユーザの割込み処理プログラム名及び処理プログラムから復帰した時のプログラム・ステートの遷移を定義する。

その表現形式は，

ＯＮ　割込み要因　ＤＯ　処理ルーチン名　〔，Ｎ〕

で表わされ，Ｎはステート遷移を示すステート番号である。

(4) プログラム・ステートの定義の終了を示す。

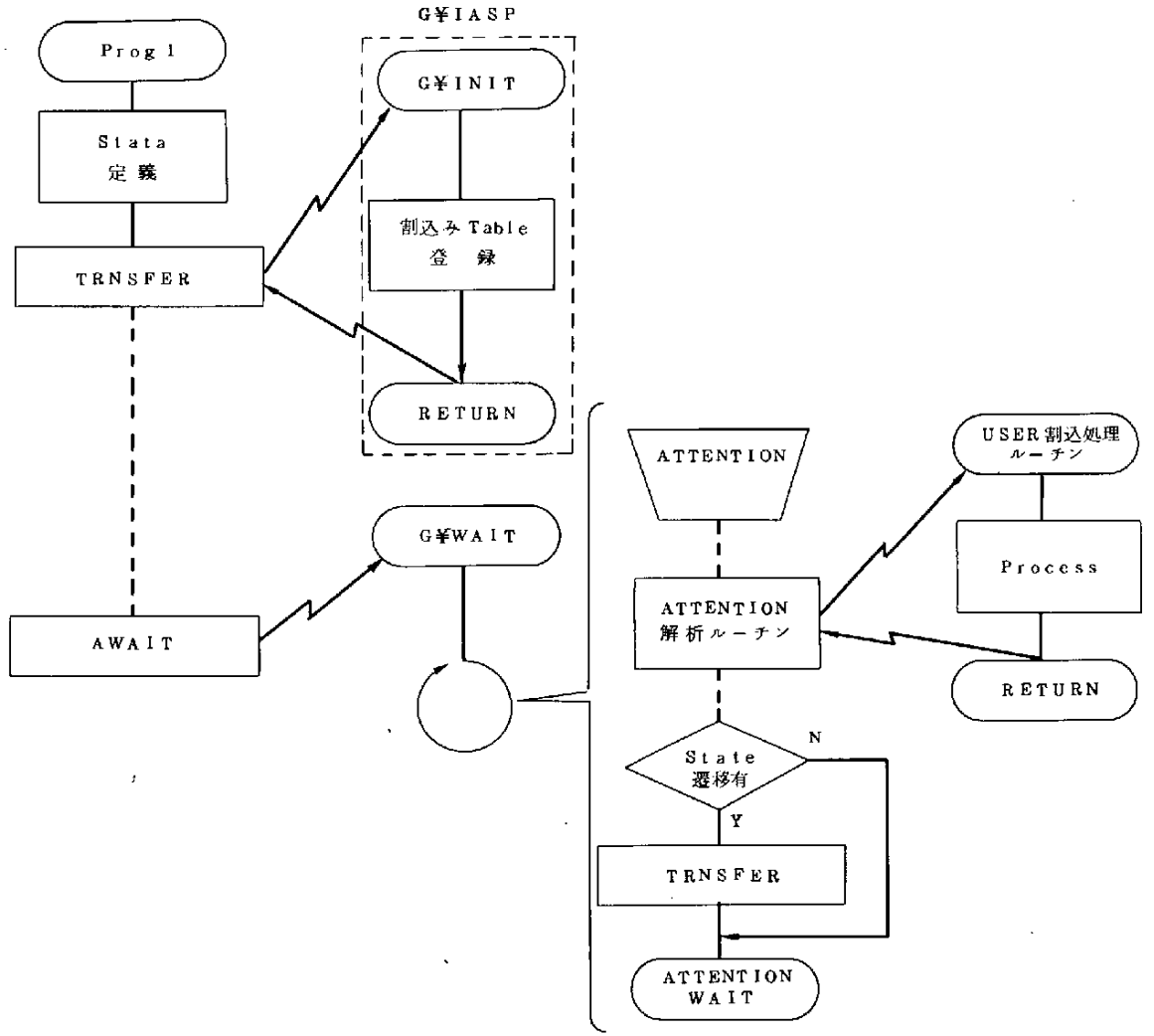
＝プログラム・ステート遷移の説明＝

上の例では，ステート１ではFKEY割込み，ステート２ではFKEY，LPEN割込み，ステート３ではMESG，TRCM割込みがそれぞれ可能である。それまでに決定されたプログラム・ステートの中で，指定された割込み要因があると，ユーザの割込み処理プログラムを実行する。さらに復帰後のステートの定義があれば，そのステートに移動する。例えばステート２においてFKEYの割込みがあれば処理ルーチンSUB2を実行した後にステート３に移る。又LPENの割込みがあれば処理ルーチンSUB3を実行するが，その後のステートは２のまゝである。

以上がプログラム・ステートの定義と，スタティックに定義されたプログラム・ステートの遷移を述べたものである。

〔図４－６〕にプログラム・ステートの遷移のアルゴリズムを示す。

〔図4－6〕　プログラム・ステートの遷移

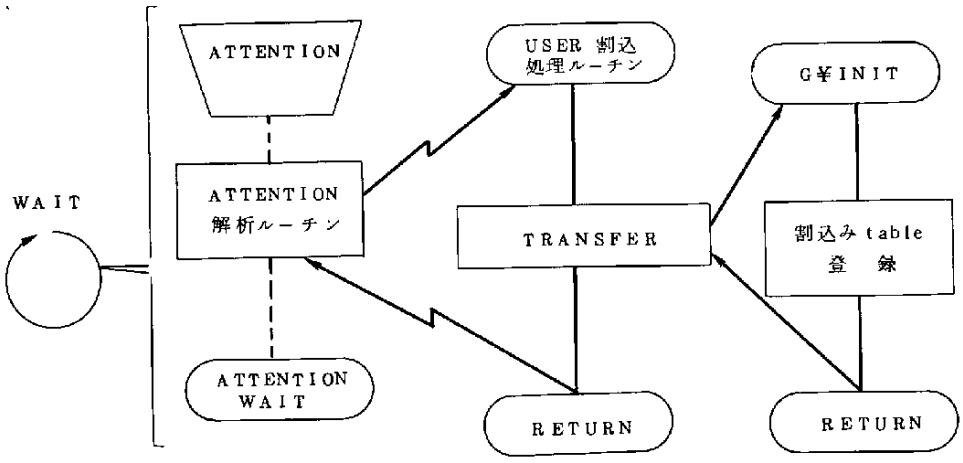


4.4.2 プログラム・ステートのダイナミックな遷移

割込み処理過程を表現するのに，プログラム・ステートの遷移を考えたが，次にダイナミックなプログラム・ステートの遷移について述べてみる。

ダイナミックなプログラム・ステートの遷移はTRANSFER文によってなされる。これは，プログラム・ステートの遷移が実行時に動的に決定される場合には必ず必要とされる。又，ユーザ・プログラムがイニシェートされた時には，プログラム・ステートはシステム・ステートに存在し，あらゆる割込み要因に対して受入れ可能な状態にある。この状態を抜け出すためには，ユーザはTRANSFER実行文を出して，プログラム・ステートをどの状態にさせるかを決定する必要がある。又，ユーザの割込み処理ルーチンの中でTRANSFER文を実行すると，スタティックに定義されたプログラム・ステートの遷移よりも優先度が高く，復帰後のステートは，TRANSFER文のステートに移る。この様子を〔図４－７〕に示す。

〔図4－7〕 プログラム・ステートの動的な遷移



4.4.3 プログラム・ステートの内容変更

プログラム・ステートに関する情報は，実行に先だってスタティックに定義され，プログラム・ステートテーブルに登録されるが，このステートの内容を動的に変更する事が可能である。これはプログラム・ステートの遷移を考えない割込み処理過程を表現する時に便利であり，ユーザの割込み処理プログラムで変更されるのが一般的である。この機能においては，指定されたステート内の割込み要因，処理ルーチン，ステート遷移情報などを変更する事が可能である。

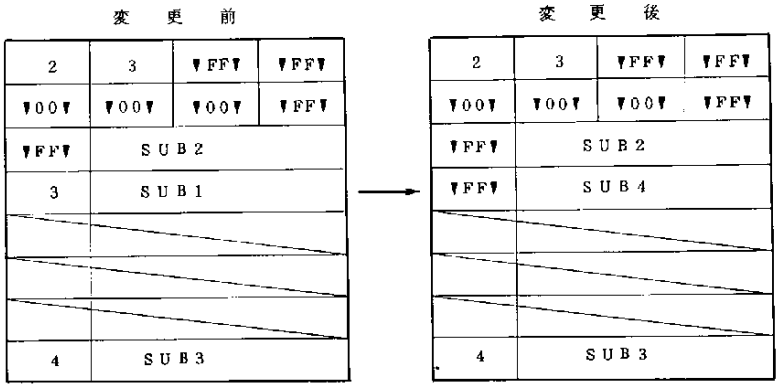
プログラム・ステートの内容変更に関するステートメントにはON変更文があり，その表現形式は，

ON 割込み要因 DO 処理ルーチン名，〔N〕，M

で表わされる。このON変更文は，スタティックな定義に用いられるON定義文の形式に，Mが追加された表現形式を持ち，Mは変更するステートの番号を表わす。この例を〔図4－8〕に示す。

ステート定義文	ステート変更文
STATE 2	ON LPEN DO SUB4, 2
ON LPEN DO SUB1, 3	
ON FKEY DO SUB2	
ON MESG DO SUB3, 4	

〔図4－8〕 ステートの内容変更



4.5 割込み待ちと割込み解析

前にも述べた様に，同期処理を用いるユーザは，プログラムを割込み待ちの状態にする事が必要であるが，そのためのステートメントとしてAWAIT文と，ATAKE文の２つのステートメントが用意されている。それらの処理について説明する。

4.5.1 AWAIT 文

プログラムをシステムの持つアイドルループに存在させ，ＣＲＴからの何らかのアクションがある迄その状態を保持している。このAWAIT文は割込み処理プログラム以外のプログラムで実行されるのが普通であるが，割込み処理プログラムで出されてもさしつかえない。しかしその割込み処理プログラムのリカーシブルコールに注意を要する。又多重の割込みの場合には，AWAITが出されて現在割込まれているプログラムを呼ぶ事のない様に注意する必要が生じる。

通常は，プログラム・ステートの概念を用いる事と次に示されるATAKE文を利用する事によって多重割込みと同様の処理を行えるので，割込み処理プログラム内では，AWAIT文は必要とされない。

一度AWAITが実行されると，割込み処理プログラムのRETURN文のみでは，システムのアイドル・ループに入って，割込み待ちの状態になる。この状態を脱出したい時には，割込み処理プログラム内でAEXIT文を実行する。この事によって，プログラムの制御は一番最近に出されたAWAIT文の次のステートメントに渡る。

4.5.2 ATAKE 文

ATAKE文は，次の形式で表わされる。

ATAKE　変数名，　割込み要因，〔，割込み要因・・・〕

この文が実行されると，システムは割込み待ち状態に入り，アテンションを受けつける状態にある。何らかの割込みが入ると，割込み要因の解析を行ない，該当するものであると，割込み要因の記述された順序数に従って，変

数名に，１～６迄の正変数が与えられる。割込み要因が該当しない場合には0が与えられる。この後，プログラムは，システムの割込み待ち状態から脱出し，ATAKE文の次に移る。このATAKE文は，割込みプログラム，割込み処理プログラム以外のプログラムのどちらで指定されても効果のあるもので，割込み要因の解析のみを行ない，それ以後のコントロールはユーザに任せてある。すなわち，割込み制御表でシステムにコントロールされ，割込み処理プログラムに自動的に制御を渡す方法と異なり，ユーザは割込み処理用プログラムを任意に呼び出す事が可能となっている。

4.5.3 割込み制御

UNGLの割込み制御は，ステートに関する割込み要因の抑制，解除と，ステートを意識しない割込み要因の抑制，解除とにわけられ，次のステートメントによって実行される。

(1) 抑制の時

LOCK　割込み要因〔，N〕

(2) 解除の時

UNLOCK 割込み要因〔，N〕

これらのステートメントは，割込み処理プログラム内でも，割込み処理プログラム以外のプログラム内でも実行可能である。

Nを定義した時には，ステートを意識したものであり，指定されたステートの定められた割込み要因に対してディザーブルのマスクをかけたり，イナーブル（enable）にしたりする。

4.6 割込み情報の内容

ユーザが何らかの割込み要因で割込んだ場合，その割込み情報は，プログラム上に反映しなくてはならない。すなわちプログラムが割り込み情報の内容を必要とする場合にはいつでもそれに応じられなければならない。各割込み要因に対する割込み情報の内容を〔図４－９〕に示す。これらの割込み情報はCGOSのGCHから転送されてくるものである。この情報の利用方法について簡単に述べてみる。例えば，トラッキングマークを意図とする位置まで動かし，TRCM割込みを出すと，ユーザの指定した領域に〔図4－9〕に示したトラッキング情報が入る。この情報即ちXPR，YPRによってCRT上のXY平面上の一点が定まり，ユーザ・プログラムはそれによって何らかの操作を行なう事が出来る。

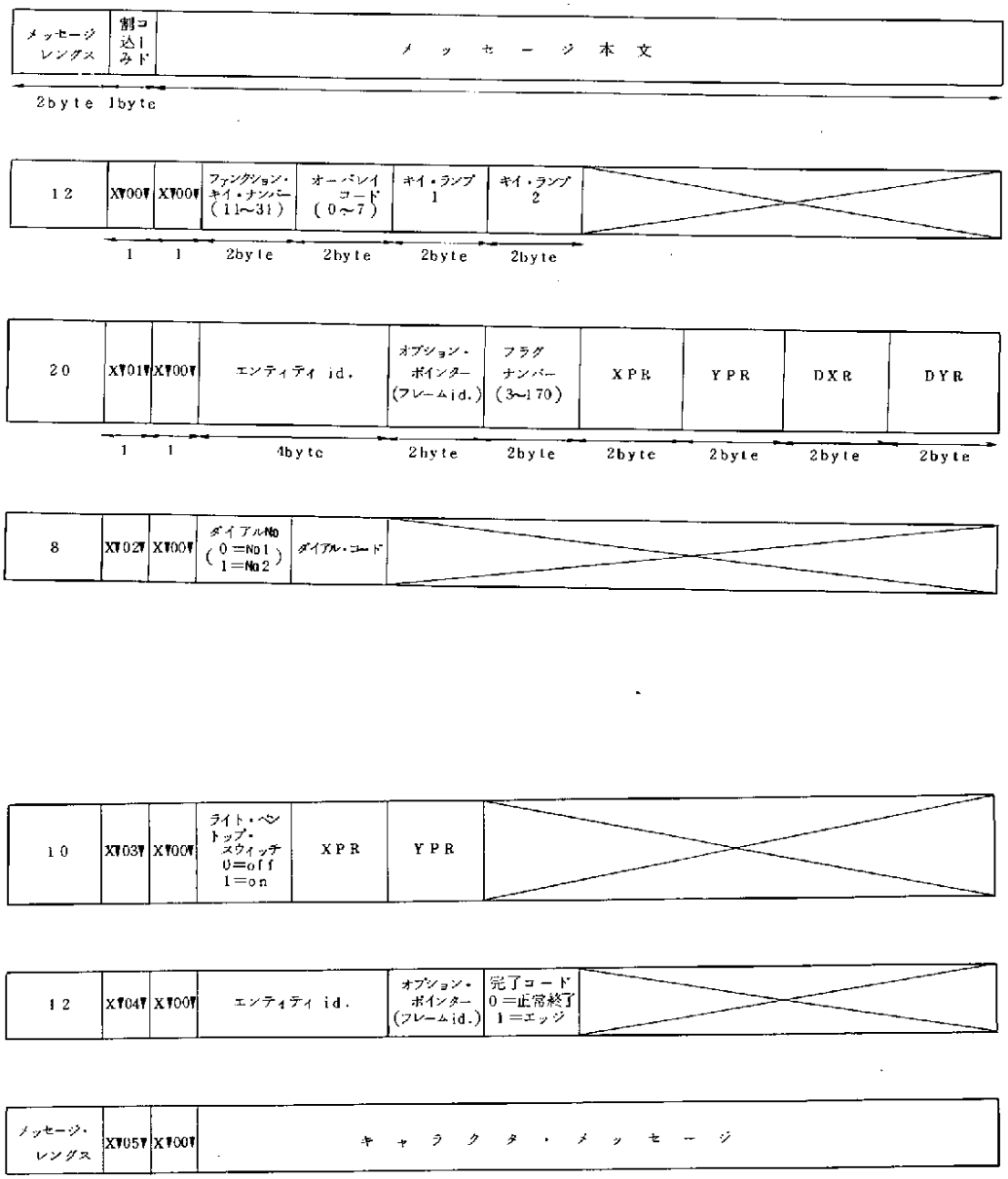
この割込み情報に関しては，ユーザが任意の領域に作成することができるが，この領域を指定しないと，ユーザは，割込み情報の検索を行なう事が不可能である。

割込み情報の内容を知るためのステートメントは次の通りである。

SPECIFY　　配列名

これ以前にユーザは，配列名で指定される配列を宣言しておく必要がある。又この配列はユーザの各割込み処理プログラムで共通に使用した方が効果的であり，そのためには，COMMON宣言しておく事が望ましい。

〔図４－９〕 割込み情報の内容



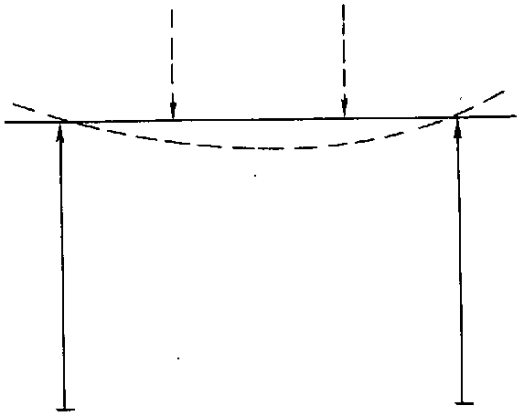
4.7 割込み処理のプログラム例

割込み処理を頻繁に行なうプログラムを考えると，プログラム・ステートによる表現，ON実行文による表現，ATAKE文による表現等が考えられられる。ここで具体的に〔図４－１０〕の様なモデルを考え，インタラクティブ処理のプログラム例を示す。この例は，モデルの上から荷重をかけた時のたわみの状態を求める。

＝手　順＝

① モデルの表示
② 荷重位置の表示
③ 荷重位置の移動
④ 荷重位置の固定と荷重値のセット
⑤ たわみの表示
⑥ モデルの再表示
⑦ メイン・プログラムへの復帰

〔図４－１０〕 割込みモデル例

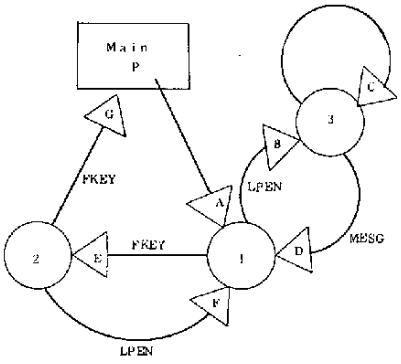


A．プログラム・ステートによる表現

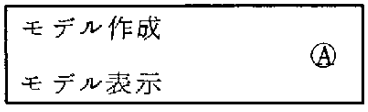
プログラム・ステートによる表現を考える時の，ステートダイヤグラムを〔図４－１１〕に示す。

なおプログラム例でⒶ～Ⓕは，ステート・ダイヤグラムのリアクションに対応する。

〔図４－１１〕 ステート・ダイヤグラム



```
MAIN PROGRAM
DIMENSION SPEC(20)
DEFINES
 STATE 1
   ON   LPEN   DO SUBB, 3
   ON   FKEY   DO SUBE, 2
 STATE 2
   ON   LPEN   DO SUBF, 1
   ON   FKEY   DO SUBG
 STATE
   ON   LPEN   DO SUBC
   ON   MESG   DO SUBD, 1
 ENDS
```




```
 SPECIFY  SPEC
 TRANSFER 1
 AWAIT

 STOP
 END

 SUBROUTINE  SUBB
 X=SPEC(2)
 Y=SPEC(3)
```

```
荷重位置を表示                    Ⓑ

      RETURN
      END

      SUBROUTINE   SUBC
      X = SPEC(2)
      Y = SPEC(3)

荷重位置を移動表示                Ⓒ

      RETURN
      END

      SUBROUTINE   SUBD
      G = SPEC

荷重値を計算
荷重値表示                        Ⓓ

      RETURN
      END

      SUBROUTINE   SUBE

たわみの計算
たわみの表示                      Ⓔ

      RETURN
      END
```

```
SUBROUTINE   SUBF
```

モデルの再表示 （荷重を除く）	Ⓕ

```
RETURN
END

SUBROUTINE   SUBG
AEXIT
RETURN
END
```

B．ＯＮ文で表現

```
MAIN   PROGRAM
DIMENSION SPEC(20)
SPECIFY   SPEC
```

モデル作成 モデル表示

```
ON   LPEN   DO   SUBB
AWAIT
ON   LPEN   DO   SUBC
ON   MESG   DO   SUBD
AWAIT
ON   FKEY   DO   SUBE
AWAIT
ON   LPEN   DO   SUBF
ON   FKEY   DO   SUBG
AWAIT
GO   TO   100
STOP
END

SUBROUTINE   SUBB
X = SPEC(2)
Y = SPEC(3)
```

荷重位置を表示

```
      AEXIT
      RETURN
      END

      SUBROUTINE   SUBC
      X = SPEC(2)
      Y = SPEC(3)
```

荷重位置を移動表示

```
      RETURN
      END

      SUBROUTINE   SUBD
      G = SPEC
```

荷重値を計算
荷重値表示

```
      AEXIT
      RETURN
      END

      SUBROUTINE   SUBE
```

たわみの計算
たわみの表示

```
      AEXIT
      RETURN
      END

      SUBROUTINE   SUBF
```

モデルの再表示
（荷重を除く）

```
      AEXIT
      RETURN
```

```
      END

      SUBROUTINE   SUBG

      STOP
      END
```

C. ATAKEによる表現

```
      MAIN   PROGRAM
      DIMENSION   SPEC(20)
      SPECIFY   SPEC
```

モデル作成 モデル表示

```
100   ATAKE   I, LPEN ,FKEY
      GO   TO   (200,600),I
      GO   TO    100
200   X = SPEC(2)
      Y = SPEC(3)
```

荷重位置を表示

```
300   ATAKE   I, LPEN ,MESG
      GO   TO   (400,500),I
      GO   TO    300
400   X = SPEC(2)
      Y = SPEC(3)
```

荷重位置を移動表示

```
      GO   TO   300
500   G = SPEC
```

荷重値を計算 荷重値表示

```
      GO   TO   100
```

600 たわみの計算
　　たわみの表示

```
700   ATAKE   I, LPEN, FKEY
      GO   TO   (800, 900), I
      GO   TO    700

800   モデルの再表示
      （荷重を除く）

      GO   TO    100
900   STOP
      END
```

5. データ・ストラクチャ操作

5. データ・ストラクチャ操作

5. 1 Associative Processingに関して

コンピュータ・グラフィックスにおいては，図形データの構造化，問題のモデリングと関連して，データ相互間の関連を定義したり，あるいは相互に関連づけられたデータ集合の中から特定の項目を検索したりする必要性がおきてくる。この様にデータ間の関連を扱う一連の処理を一般にAssociative Processing等と呼んでいる。Associative Processingは，グラフィックに限った話ではなく広くアーティフィシャル・インテリジェンス，オペレーティング・システム，CAD，IR等の分野でその処理の基本となっているが，グラフィックスにおいては特にその処理上の特徴として次の様な性格が要求される。

(1) インタラクティブ・システムとして成り立つためには，その処理に即時性が要求されること。

(2) 対象とする任意の問題をモデル化することを可能にするには，かなり一般性のある関連の表現ができなくてはならない。

(3) 扱うデータ量に関しても一般性が望まれる。

この様な要求を満たす目的からこれまでに各種のデータ構造が考えられてきた。それらは一般に構造化の原理から大きく2つに分類される。即ちリスト処理を基本とするリング・ストラクチャとハッシュ技法を基本とする連想記憶機構である。

リング・ストラクチャは，上記の要求にかなりよくかなうものとして現在も各種のアプリケーションにとり入れられ広く利用されている。しかし効率的にデータ・ストラクチャを編成するためには，扱う側からその構造をかなり意識する必要がある。またデータ量が増えて2次記憶に領域を拡張する必要が生じるとその処理効率が著しく低下するといった問題が指摘されている。ページ化に関して特に解決を与える方向としてハッシュ技法を基本とする連想記憶機構が考案された。これは関連を表現するのに順序3つ組（トリプル）

を基本とする方法で，３つ組を構成する２項のダブル・ハッシングにより定まるアドレスに３つ組を記憶することにより関連を保持する。この時ダブル・ハッシングを行う２項はそれぞれアクセス・ワード，チェック・ワードと呼ばれる。同じアクセス・ワードを持つ３つ組を同一頁にマッピングする様にすると，アクセス・ワードが決まればアクセス・ページが決まることになるから，検索時には原理的に１回の２次記憶アクセスで目的とする関連（３つ組）をとりだすことができることになる。このタイプのデータ・ストラクチャを始めて使用した言語としてLEAPが有名である。

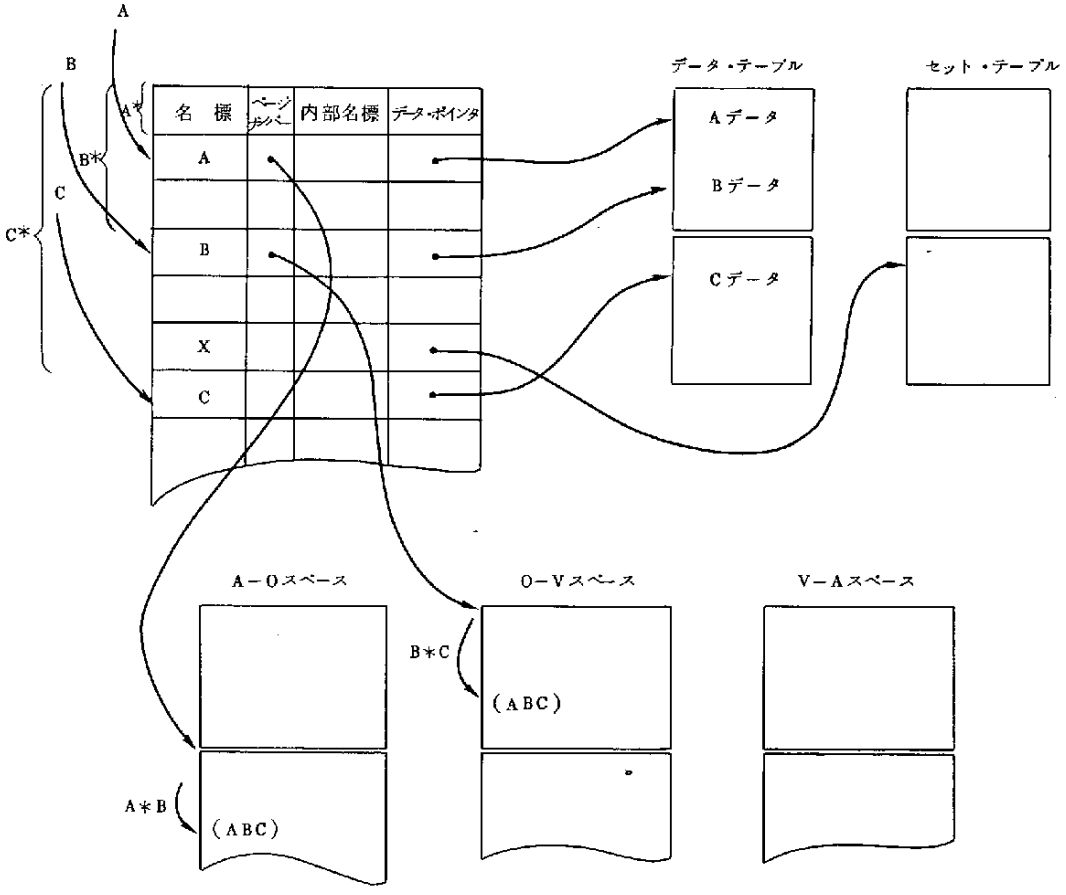
LEAPは１９６７年Lincon Laboratoryで開発されたもので，そのインプリメントはコンパイラ・ジェネレータVITALが使われたという。現在もグラフィックス，ＡＩの分野で効果的に利用されている。その他にもLEAPに類似したシステムが各所で開発されている。

UNGLの扱うデータ構造もLEAPのそれにほぼ近いものであるが，機能的にSETに関する扱いをやや強化してある。

5．2　構造の特徴及び原理

〔図5－1〕は記憶機構の概念図を示したものである。以下構造の概略とアクセスのメカニズムについて簡単に原理を説明する。

〔図5－1〕 連想記憶の機構



（注）A*はAにハッシング操作を与えること

A*BはA，Bでダブル・ハッシングを行うこと

5．2．1　ディクショナリ・テーブル

関連づけの基本となる要素はアイテムと呼ばれ，プログラム上ユニークな名標を与えられる。アイテム名標は6文字までの文字ストリングとして与え

られるが，このままでは領域を浪費し内部的な扱いに不都合があるため，適当にコンパクションして内部名標に変換する必要がある。ディクショナリ・テーブルはアイテム名標と内部名標との対応を与えるものである。ディクショナリ・テーブルの格納位置はそのアイテムをハッシングすることにより決定されるから，内部名標への変換はハッシング操作によって高速に行われる。内部名標の下位ビットは適当にユニークに決め，その上位ビットはアクセス・ページ（連想テーブル）を表わす様にしておく。この様にすれば任意のアイテムが与えられた時，それをアクセス・ワードとするトリプルを格納するページがその上位ビットから知ることができる。またそのアイテムのデータ部が格納されているデータ・テーブルへのポインタもテーブルに格納されている。

5．2．2　連想テーブル

連想テーブルにはトリプル（A，O，V）が格納される。A－Oスペース，O－Vスペース，V－Aスペースの3つのスペースから成り，トリプルが定義されるとその各項をアクセス・ワードとして3通りの方法で重複して3領域に各納するようにする。

例えば（X，Y，Z）の3つ組が定義されると，Xをアクセス・ワード，Yをチェック・ワードとして（X，Y）でハッシングを行い，トリプルのマッピング・アドレスを決定しA－Oスペースに格納する。同様に（Y，Z），（Z，X）でO－V，V－Aスペースにマッピングしその位置にトリプルを格納する。この様にすればトリプルのどのアクセス項が与えられても，それに対応するページをアクセスすることにより効率の良い検索ができることになる。

この様にして構造化された機構に対して一般に次の形式の質問が可能になる。これはSAF等と呼ばれる。

$$F_1 \qquad X\ (Y) = Z$$

F_2　　X (Y) = ?

F_3　　X (?) = Z

F_4　　X (?) = ?

F_5　　? (Y) = Z

F_6　　? (?) = Z

F_7　　? (Y) = ?

F_8　　? (?) = ?

F_8 は特殊なケースで連想テーブルの全ダンプを意味する。2項によるダブル・ハッシングの結果マッピング・アドレスを決定するわけであるが，この時次の2つの例外ケースに注意する必要がある。1つは異なる2項同志たとえば（X，Y）と（U，V）のマッピング・アドレスが全く一致する様な場合で，これはコンフリクトと呼ばれる。コンフリクトが起きた場合の処理としては，一般にそのセルにコンフリクト表示をしそこから別の領域（コンフリクト用領域）にリストをつけるか，またはコンフリクトを起した場所を起点としてさらにある操作を加え，次のマッピング・アドレスを見つける方法である。どちらの方法も通常のハッシュ・テーブルにおけるコンフリクト処理と原理的に全く同じものであるが，連想記憶では後述するマルチプル・ヒットの様なケースも扱わなくてはならないからその処理はさらに複雑になる。リストによる解決では，リスト処理がその処理の大部分を占めるという様なことがないように配慮する必要がある。また後者の方法ではマルチプルヒットの場合も考慮すると，処理アルゴリズムが複雑化し過ぎる恐れがある。

5．2．3　データ・テーブル

データ・テーブルはアイテム固有のデータを格納するテーブルである。あるアイテムが定義されると，そのアイテムに関するデータを格納するブロックがデータ・テーブルに確保され，ディクショナリ・テーブルにそのアドレスが登録される。従って検索はディクショナリ・テーブルからブロック・アドレスを知り，そのデータにアクセスするという過程で行われる。一般にア

イテムの保有するデータ量は，アイテムによって異なると考えられるから，データ・ブロックは不定長ブロックとして管理しなくてはならない。ストレージ管理を効率的に行うためには，データ・ブロックのタイプをいくつか定め，め，各タイプ毎にストレージを編成するのが効果的である。

5. 2. 4 セット・テーブル

セット・テーブルはセットのメンバーであるアイテムの集合を蓄えている。セットもアイテムと同様にディクショナリ・テーブルに登録される。セットの場合アイテムのデータ・ポインタにあたる部分にセット・テーブルへのチェインが格納されている。セット・テーブル内には複数個のセットが格納されているが，各ページ毎にヘッダ・リストを保持していてヘッダ・リストをたどることにより，目的とするセットのメンバー・チェインの入口が分る様になっている。メンバー・チェインはそのセットのメンバーとなるアイテムをチェインしたもので，処理の都合上内部名標で順序化されている。しかし扱う側からはunordered setとして考えなくてはならない。つまりメンバーの挿入位置等を外部から制御することはできない。

E，アクセス法

データ・テーブル，セット・テーブルに関してはその構造の説明の中でアクセスのメカニズムの概要についてふれた。ここでは連想テーブルのアクセス法に関して例を中心に簡単に説明を加える。

(1)，トリプル（X，Y，Z）の創成

X，Y，Z（各々文字ストリング）をそれぞれハッシングすることによりディクショナリ・テーブルへの登録アドレスを求め，その位置に名標を登録する。また内部名標を決定する。まずA－Oスペースに対しては，内部名標の上位ビットで定められたアクセス・ページ内の，X，Y 2項のダブル・ハッシングで定まるアドレスにトリプル（X，Y，Z）を格納する。同様にO－V，V－Aスペースにトリプル（X，Y，Z）をそれぞれ格納する。

(2)，トリプルの検索

創成の時と全く同じ操作でトリプルを検索することができる。例えばX・Y＝？の質問に対しては，まずX，Yをディクショナリ・テーブルから対応する内部名標に変換し，Xの内部名標からA－Oスペースへのアクセス・ページがわかる。そのページ内でX，Yのダブル・ハッシングから定まるアドレスをアクセスすれば目的とする（X，Y，Z）を発見することができることになる。

5．3　問　題　点

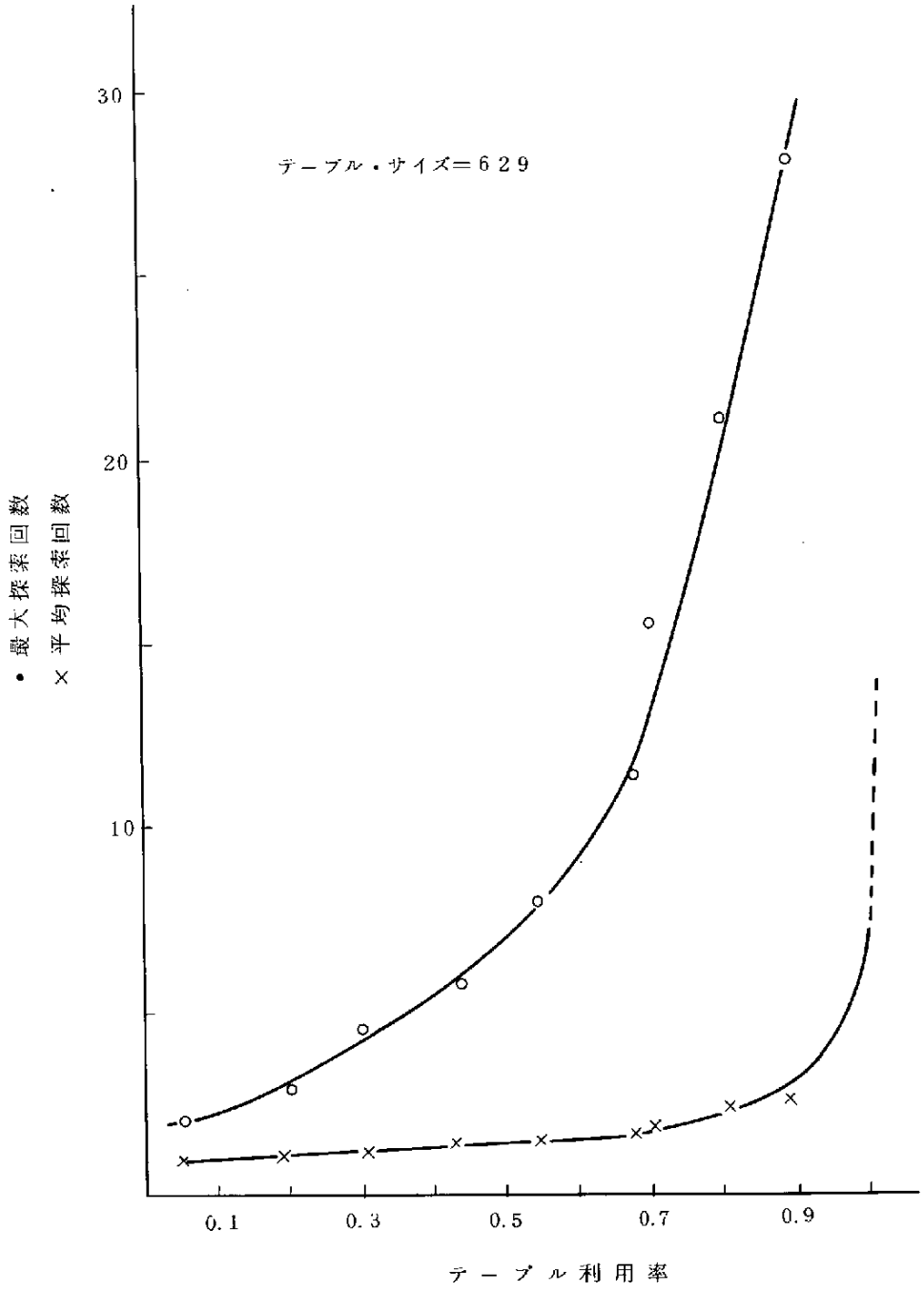
5.1，5.2で述べたように，このタイプのデータ・ストラクチャは，関連の表現力，検索能力，2次記憶スペースへの拡張性においてかなり優れた力を発揮するが，先にも述べたコンフリクト等の問題も含めて，内部的に検討されるべき問題がいくつかある。この項では実際にインプリメントにあたって，特に検討の対象となった問題点について簡単にふれ，その解決の方向の概略について説明する。

(1)，コンフリクトの問題

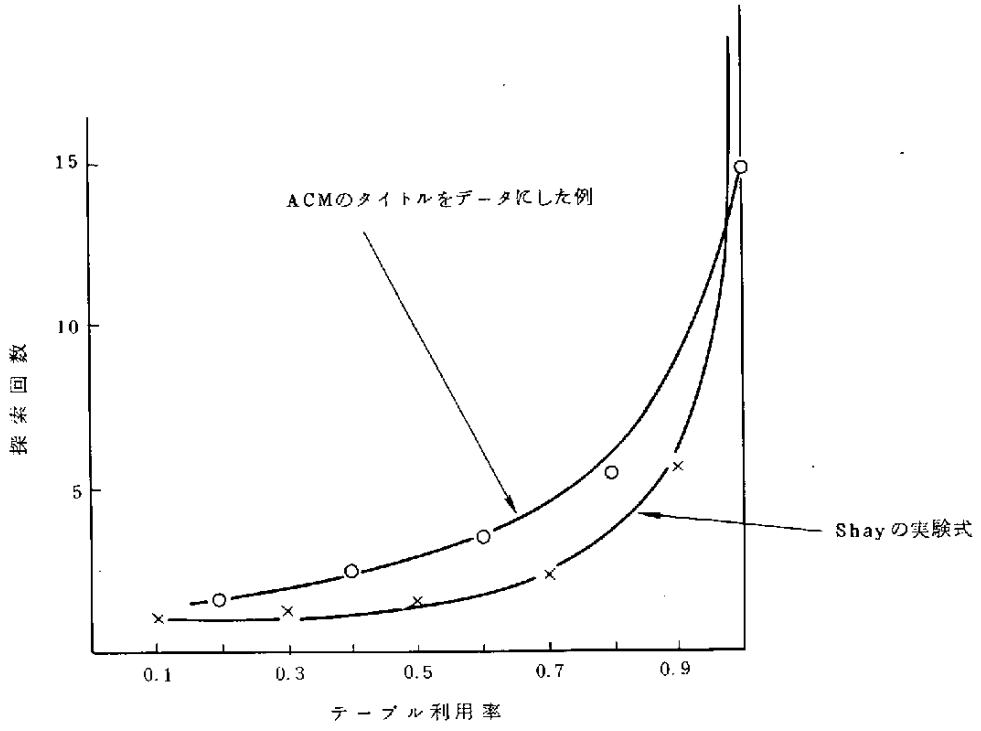
コンフリクトが頻繁に発生すると，その処理に多くの時間を費し探索効率を落す結果となる。コンフリクトの問題はディクショナリ・テーブル，連想テーブルの両者について検討を加える必要がある。

ディクショナリ・テーブルにおけるコンフリクト及びテーブル利用率の関係については，実験的に次のようなカーブが得られた。〔図5－2〕またShay等の実験結果（〔図5－3〕）もこれにほぼ近い結果を示している。

〔図5－2〕 実 験 結 果



〔図5－3〕　Shay の求めた結果



この結果から，テーブル利用率が８０～９０％附近から急激に探索回数が上昇することがわかる。つまりこの附近から急激にコンフリクトが増え初めるわけである。従って効率的な検索を行うためには，テーブル利用率が７０～８０％ぐらいになった時にそのテーブルの使用を中止するように考慮する必要がある。

連想テーブルにおけるコンフリクト処理は，また別の問題を含んでいる。すなわち2項によるダブル・ハッシングというハッシングのメカニズム自体の違いからくる操作上の困難さとマルチプルヒットという別の事態と関連し

た処理の複雑さである。この様な処理の複雑化を避けるためには，コンフリクト処理の仕方としてリストによりコンフリクト・セルを別領域に移し，リスト構造で管理する方法がより効率的であることがわかった。しかしこの時もリスト処理が主流を占める様になっては，このタイプの構造のメリットが失われることになるから，その領域の配分には十分に注意を払う必要がある。

(2)，A－O，O－V，V－A各スペースにおけるアクセス・ワードの偏りについて

A－Oスペース，O－Vスペース，V－Aスペースへのトリプルを格納する時，1つのアクセス・ワードは必ず同一のページに属するという原則のもとに，一般に一ページに複数個のアクセス・ワードが収容される。この様にして実際にトリプルが格納されていく過程を考えると，トリプル自体の記述は3領域に対して対称であるが，プログラムの総体をみた時それが対称であるという保証がないため一般にあるアクセス・ワードに関するトリプルの分布がページ毎に偏りを生じる。従って無計画にページにアクセス・ワードを割り当てていくと，急激にページ内のトリプルが増大しページ・オーバーフローを招くことになる。しかしあるアクセス・ワードに関してどの様な形で関連（トリプル）が定義されていくかはプログラムが実際に進行しなくては分らないから，前もってその割り付けを制御することはできない。またページ毎のページの状態情報を主記憶に蓄えるには全ページ分の管理情報を置くことになり領域の浪費を招くことになる。このためアクセス・ワードをページに割り当てる適当な方法を考えなくてはならない。折衷的な方法として数ページのページ管理情報を主記憶に置き，プログラムの進行する過程でトリプルの生成を監視し，なるべく平均的にトリプルが格納されるようにアクセス・ワードをページに割り当てる方法が効果的である。

(3)，ページ内にトリプルが格納し切れなくなった時，すなわちページ・オーバフローにどう対処するか

ページ・オーバフローが発生するとそのページにあるトリプルをすべて別

のより大きなページにマッピングし直すか，もしくはそのページに続くオーバフロー・ページに新たに関連を定義していく等の方法を講じなくてはならない。これは処理効率を低下させる大きな原因となる。これにはページ・オーバフローを発生する以前にアクセス・ワードをページ毎の偏りがないようになるべく均等に割り当てることが必要である。しかしページ・サイズは主記憶の関係上あまり極端には大きくできないから，特殊な場合にはページ・オーバフローは避けられないこともある。またページ・サイズの決定は一般的な関連処理の統計から適当に決定することが必要である。

(4)，トリプルを生成したり，消去したりする時，3領域にわたってページ・アクセスするのは非効率である。

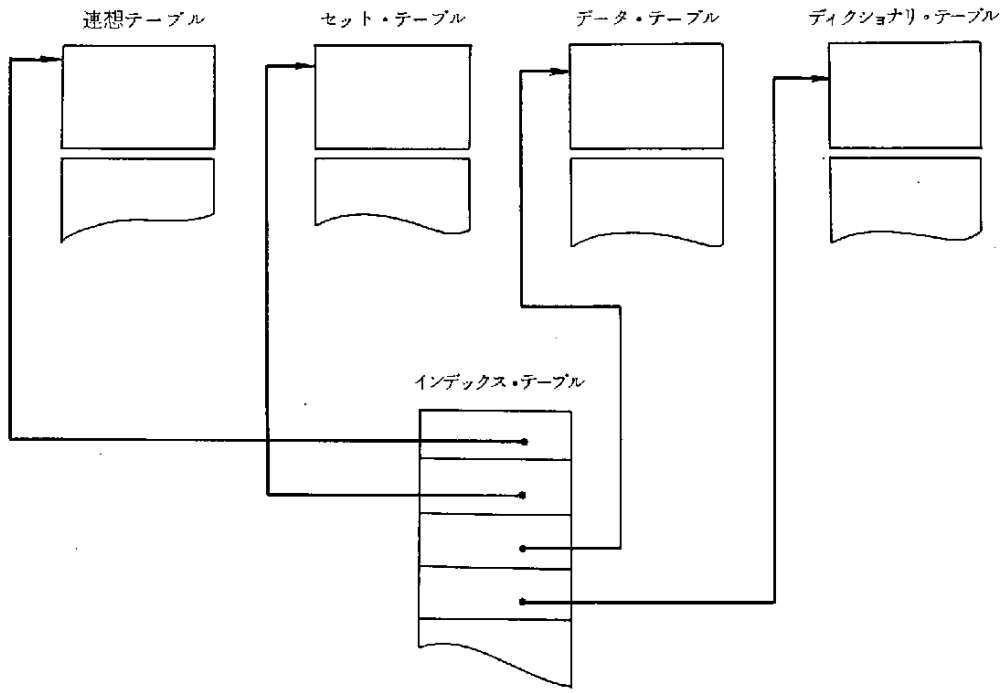
これにはトリプルの生成，消去を直接的に行わず，一担主記憶内のコマンド・リストに登録を行い，実際にページが主記憶にロードされた時にそのページに関する全要素の書き込み，消去を行うようにすれば効率があがる。

5．4　各ページの構造と管理方式

5．4．1　ページ化の対象について

ディクショナリ・テーブル，連想テーブル，セット・テーブル，データ・テーブルはすべてページ化され，主記憶中に各々必要とされる１ページだけが読み込まれている。また各ページの読み込みアドレスを管理するインデックス・テーブルが主記憶中に置かれ，現在どのページが主記憶中にあるか等も併せて管理している。

〔図５－４〕　各ページの関係

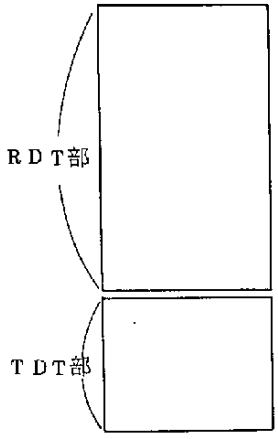


5. 4. 2　各ページの構造と管理方式について

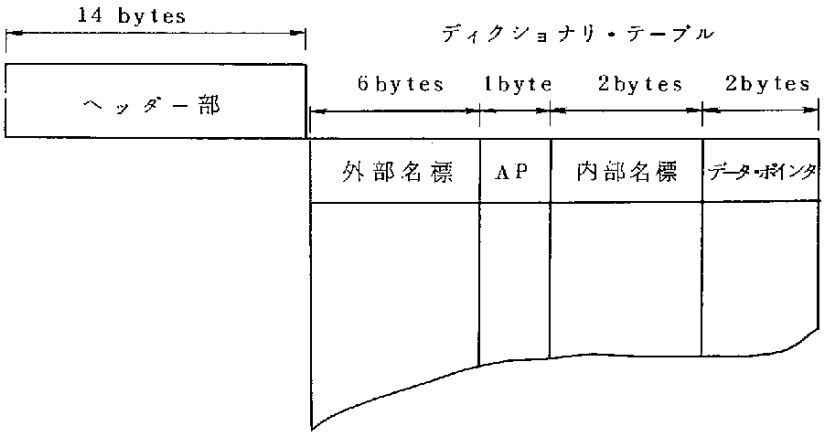
A，ディクショナリ・テーブル

ディクショナリ・テーブルは主記憶に常駐する部分ＲＤＴ部とページ化されて扱われる部分ＴＤＴ部から構成される。アイテムの登録は最初ＲＤＴ部に対して行われるが，ＲＤＴ部がオーバフローを起すとＴＤＴ部に対して割りつけが行われる。このアクセス・アルゴリズムの詳細に関しては後述する。〔図５－６〕はディクショナリ・テーブルのページ・イメージを示したものである。

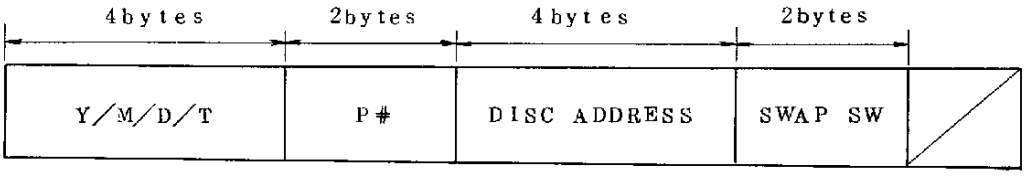
〔図５－５〕　ディクショナリ・テーブル



〔図5－6〕 ディクショナリ・テーブル



ヘッダ部の詳細



(1), ヘッダ－部

ヘッダ－部は各ペ－ジに共通な部分（先頭12bytes）とペ－ジの種類によって固有な部分とからなり，そのペ－ジに関する固有な情報を蓄えている。その内容は次の通りである。

- Y／M／D／T：プログラムのロードされた時間が記録される。（創成時には創成の時間が記録されている。）これは後でこのペ－ジが現在より以前に創成されたものかどうかを判別するのに使用される。
- P#：そのペ－ジのロジカル・ペ－ジ番号が記録される。

◦ DISSC ADRESS：そのページのディスク上のフィジカルアドレスが格納される。これはページがスワップアウトされる時必要とされるディスク・アドレスを示している。

◦ SWAP SW：そのページをスワップアウトする時，そのページが読み込まれた時と同じ状態を保っていればディスクに書き出す必要はなくそのページを破壊してしまってかまわない。もし読み込まれた後に書き込みが行われていれば，スワップアウトする時にディスクに正しいコピーを残さなければならない。SWAP SWはページに書き込みがあった時それを記憶しておく。

(2)，ディクショナリ・テーブル部

ディクショナリ・テーブルは〔図5－6〕で示すように11bytesを1単位として構成され，それぞれ次の様な情報を蓄えている。

◦ 外部名標：アイテム，またはセットのユニーク名（アイテム名標，セット名標）が文字ストリングの形で格納される。

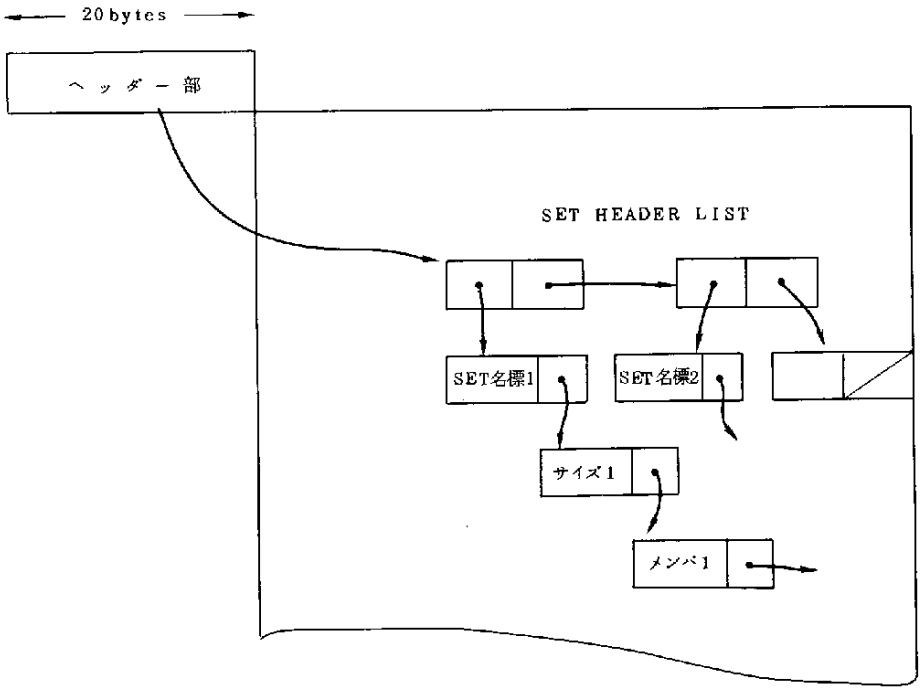
◦ P＃：そのアイテムをアクセス・ワードする時の連想テーブルのアクセス・ページ番号を格納する。アクセス・ページ番号は実際にトリプルが定義された時に，その時のページ割り当て状況をみて動的に決定される。

◦ 内部名標：対応する内部名標を保持している。内部名標としてテーブル内相対変位を与えている。従って内部名標によるテーブルのダイレクト・アクセスが可能である。

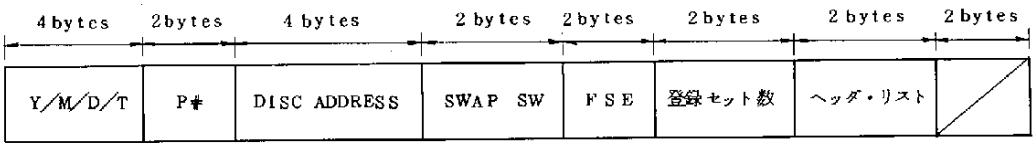
◦ データ・ポインタ：データ・テーブルのページ番号とそのページ内のデータ・ブロックのページ内の相対変位で編成される。

B；セットテーブル

〔図5－7〕 セット・テーブル



ヘッダー部の詳細



(1)，ヘッダー部

◦ＦＳＥ（フリー・リスト・エントリー）：ページ内のフリー・リストへの入口を示す。メンバを追加する時はこの中からセルを確保する。

◦登録セット個数：そのページ内に登録されているセットの種類の数を記

録している。ページの使用状況を知る時の手がかりとする。

・ヘッダ・リスト：各セットのメンバ・リストへの入口を示すセット・ヘッダのリストへの入口を示す。

(2)，セット用フリー・ストレージ

セット・テーブルは2 bytes を単位セルとして扱い，それぞれセット・ヘッダ用のセル・メンバ用のセルとして利用される。ページ内のセルは創成時はすべてのセルを一つにチェインしたフリー・ストレッヂを編成している。セットが定義されるとフリー・セルが確保され，セット・ヘッダにセット名標が登録され，そのメンバを格納したセルをつないだメンバ・チェインが創成されセット・ヘッダにつながれる。（処理の詳細についてはアクセスの項でもう少し詳しく述べる。）

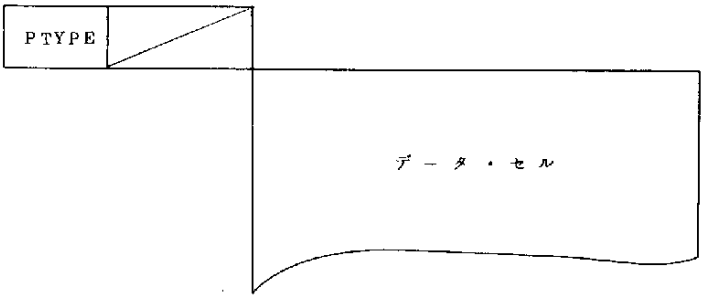
セットは原則的には1ページに1セットを割り当てるが，ページが尽きた時は，全体の使用状況をみて1ページに複数個のセットを割りあてるようにする。またページ内セットが成長しページに収容し切れなくなった時は，次のオーバーフロー処理を行う。

①，ページ内に複数個のセットが存在し，ページ・オーバフローの状態におち入った時はそのページ内で最も大きなセットを選び新たなページに移す。（新たなページとはあらかじめ設けておいたオーバフロー・ページである。）

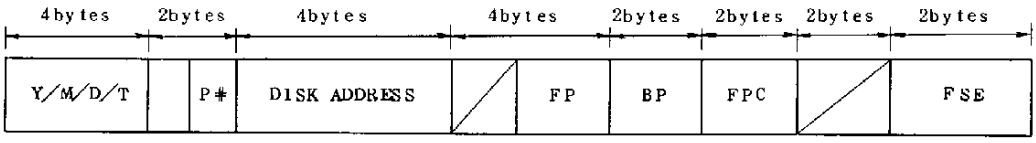
②，ページ内に1個のセットしかなく，これが生長してページ・オーバフローを起した場合はそのセットを2つに分断し2ページにわたって収容する。

C，データ・テーブル

〔図5－8〕 データ・テーブル



ヘッダー部の詳細



データ・テーブルは各アイテムの保有するデータを蓄えるが，一般にデータ長は不定長と考えなくてはならないからその管理方式を工夫する必要がある。ここではあるページに格納するデータ長は常に一定にする様な扱いをしている。

(1)，ヘッダー部

- ＦＰ（フォワード・ポインタ）：
- ＢＰ（バックワード・ポインタ）：

} 同じタイプのページをチェインする。

同じタイプのページとはその中で扱われるデータ・ブロックの長さが等しいページをいう。

- ＦＰＣ（フリー・ページ・チェイン）：これは空ページがなくなった時点で，タイプ毎に空スペースのできたページをサーチする時に使用す

る。タイプ・ページ・チェインの入口はページ・タイプ・テーブルの最後のアロケート・ページを見れば良い。

- FSE（フリー・ストレージ・エントリー）：ページ内のフリー・ストレージ・リストへの入口を示す。
- PTYPE：そのページのページ・タイプが記されている。

(2)，データ・ページのアロケーション

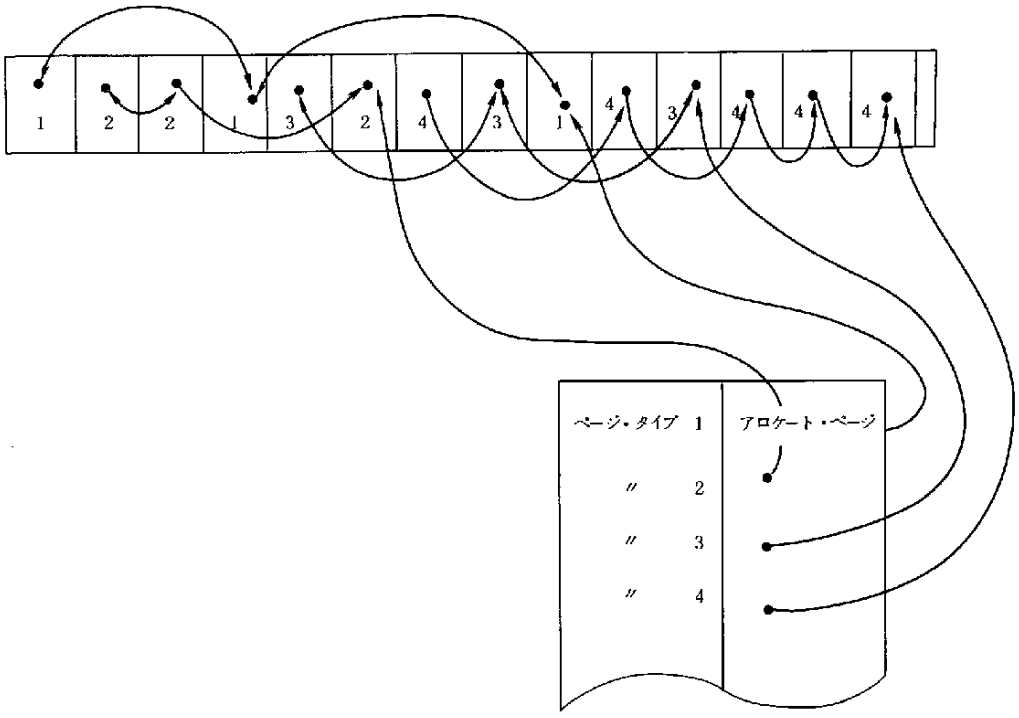
データ・テーブルのページ・アロケーションのために主記憶内に次のコントロール情報が置かれている。

- ページ・タイプ・テーブル：現在アロケード中のページとページ・タイプ・テーブルの対応表
- フリー・ページ・インディケータ：次の空ページを指示する。

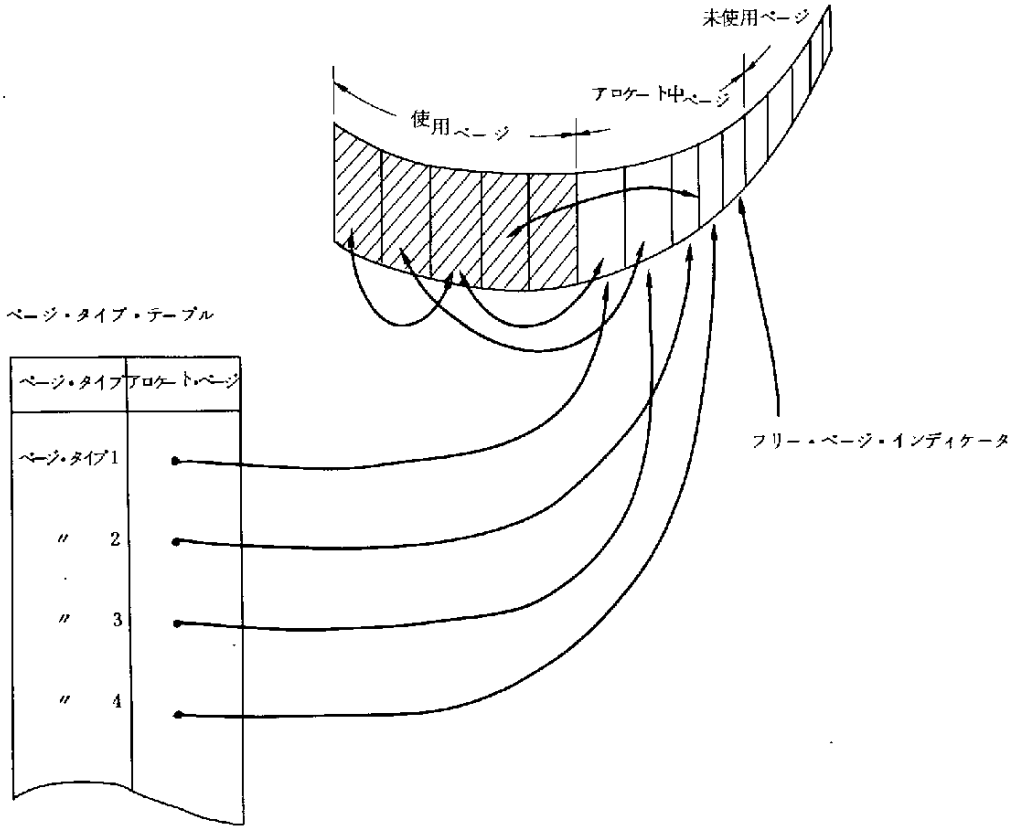
〔図5－9〕，〔図5－10〕はこれらの関係を図示したものである。例えばデータ部の長さが3のアイテムのデータ・ブロックをアロケートする時を考えると，まずページ・タイプ・テーブルからタイプ3のエントリーを見出し，その対応するアロケート・ページを知り，ヘッダ情報のフリー・ストレージ・エントリーからフリー・ブロックを確保しアイテムに割り当てる。

この時そのページ番号とそのブロックのページ内相対位置をアイテム・テーブルに登録し後の検索に備える。

〔図5－9〕 ページ・タイプテーブルとタイプページチェイン



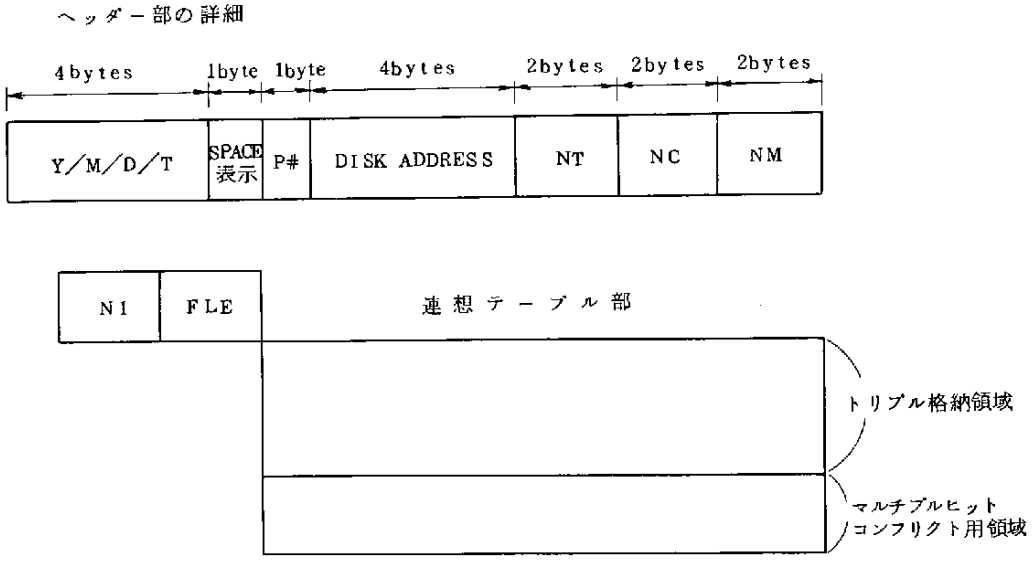
〔図5－10〕 ページ・タイプ・テーブルによる
ページアロケーション



D，連想テーブル

連想テーブルは関連表現の基本となるトリプルを格納する最も重要な役割を果たす。ここではページの構成要素に関する説明を中心にし，ページ内のセルの構成法等についてはアクセスの項で詳細を述べることにする。

〔図5－11〕 連想テーブル



(1)，ヘッダ－部の説明

- SPACE表示：このページがA－Oスペース，O－Vスペース，V－Aスペースのいずれに属するかの表示を与える。
- NT：ページ内にある全トリプル個数を記録している。アクセス・ワードのページ割り当てを行う時の参考情報となる。
- NC：そのページ内で発生したコンフリクトの回数を記録している。
- NM：そのページ内で発生したマルチプルヒットの回数を記録している。
- NI：そのページ内に登録されているアクセス・ワードの種類数を記録している。

NC，NM，NI等は直接プログラムから利用されないが，ページの状態の統計として記録され，後にページ割り当てのアルゴリズムを検討したり，スワッピング・アルゴリズムを検討したりする時の基礎データとする。

- FLE：マルチプルヒット，コンフリクト用領域のフリー・リストの入口が記されている。

(2)，連想テーブル部

連想テーブル部8バイトを一単位とするセルの集合として扱われ，セルにはトリプルが格納される。図の様にトリプル格納領域とマルチプルヒット，コンフリクト用領域の2つに分けて管理される。トリプル格納領域はハッシングの結果，直接トリプルを格納する領域である。一方マルチプルヒット，コンフリクト用領域はマルチプルヒット，コンフリクトが生じた時にセルを供給するためのフリー・セルの集りであり，リストで管理されている。すなわちトリプルをマッピングした結果その位置にすでにトリプルが登録されていれば，そのセルにコンフリクトを表示し，そのセルからコンフリクト用領域の空セルに対してリストをつけ，そこに対象とするトリプルを格納する。この詳細についてもアクセスの項で述べることにする。

ここでトリプル格納領域とマルチプルヒット，コンフリクト用領域の領域長の比をどうするかという問題を考えなくてはならない。これには適当な決定アルゴリズムが無いから，実測データをもとに効率の良い動作点を求めるのが適当である。

定性的には次の様な事がいえそうである。マルチプルヒット，コンフリクト用領域を拡張し，トリプル格納領域をしぼるとトリプル格納領域の利用度がすぐに上りコンフリクトが頻繁に起る様になる。この結果マルチプルヒット，コンフリクト用領域が使用されるようになる。この様にすると領域の利用率という点では効果があるが，コンフリクトが頻繁に発生して効率上あまり好ましくない。逆の場合，すなわちトリプル格納領域を拡張しマルチプルヒット，コンフリクト用領域をしぼると，トリプル格納領域の利用率が上

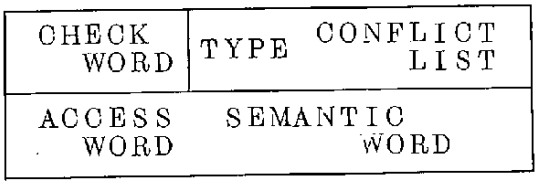
らない内にマルチプルヒット，コンフリクト用領域がコンフリクト，マルチプルヒットで一杯になりページ・オーバフローを起す可能性がある。この場合は，スペースを十分に使いきらないうちにページ・オーバフローを起すという意味でスペース利用率という点からも好ましくない。従って効率的にもスペースの利用率からも満足のいくような両者の中間的な比率を実験的に求める必要がある。現在は両者の比率を３／２に定めている。

(3)，セルの基本形

連想テーブル内のセルには次の２つの基本形がある。

セル１

〔図５－１２〕 セルの基本形 1



セル１はトリプル格納用領域及びマルチプルヒット，コンフリクト領域で扱われるセルで〔図５－１２〕の様な形で構成されている。

- ACCES WORD：アクセス・ワードを格納している。ページ内に複数個のアクセス・ワードを収容する形式をとっているので，アクセス・ワードのみによる検索に答える時にはこの項を対象にページをサーチする必要がある。
- CHECK WORD：トリプルの第２項がチェック・ワードとして格納される。マルチプルヒットか否かのチェックが行われる。
- SEMANTIC WORD ：トリプルの第３項が格納される。マルチプルヒットを起した時は，このワードにマルチプルヒット・リストがつけられる。
- TYPE：このセルの状態を表示する。

  ０の時……NORMAL

  １の時……マルチプルヒットを起している。

2の時……コンフリクトを起している。

3の時……マルチプルヒットでかつコンフリクトを起している。

トリプルの格納，検索はセルの状態を調べてから行われる。

- CONFLICT LIST；コンフリクトが発生した時，この場所にマルチプルヒット，コンフリクト用領域への空セルへのポインタがつけられる。

セル 2

〔図5-13〕セルの基本形 2

S ・P VAL1
VAL2 VAL3

セル2はマルチプルヒット，コンフリクト用領域でのみ扱われるセルで〔図5-13〕のような形式をとる。マルチプルヒットを起した時にマルチプルヒット・リストとして使用される。

- VAL1～VAL3：それぞれマルチプルヒットを起したトリプルのセマンティック・ワードが格納される。
- S：そのセルに格納されているセマンティック・ワードの個数0～3を記憶している。
- P：次のセルへのポインタを格納する。

５．５　各テーブルのアクセス・メソット

５．５．１　ディクショナリ・テーブルのアクセス法

ディクショナリー・テーブルの構成法には次の３通りの方法が可能である。

①，主記憶に常駐させる。

②，テーブルをすべてページ化し，バーチャルな部分も含めて一つのテーブルと考えてアイテムをマッピングする。

③，①と②を併用する。

①はアクセス・タイムが速くて理想的だが大きな主記憶領域を必要とするため実用化は困難である。（アイテム個数１０００で約１０ｋbytesを必要とする。）

②の方法は，アイテム個数の多少にかかわらず１－１／ｎの確立でディスク・アクセスを余儀なくされる。（ｎはページ数）この方法はアイテム数が少ない時には不利である。

以上の様な理由からここでは③の方法，即ち①の方法と②の方法を併用する形式を採用する。

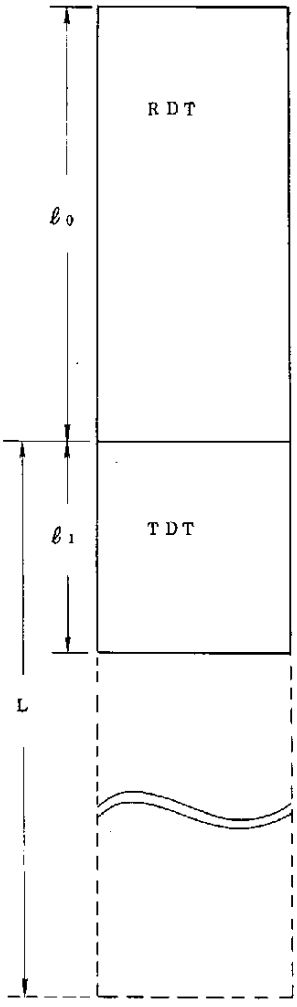
ディクショナリ・テーブルは〔図５－１４〕に示されるように主記憶に常駐するＲＤＴ部とページ化されて適宜スワップイン，スワップアウトされる部分ＴＤＴ部から成り立っている。以下アイテムの登録を例にしてそのアクセスアルゴリズムを説明する。

アイテムは始め，ＲＤＴ部にマッピングファンクションＭｏで格納される。やがてＲＤＴが満たされると，ＲＤＴへのマッピングを中止しＴＤＴに格納を始める。ＲＤＴが満たされたか否かの判定は，空セルのサーチの段階でコンフリクト回数がある一定値を越えたか否かで判定する。

又，マッピングファンクションＭｏは次の様なハッシング・メカニズムを持つ。

①，支えられたアイテム名標（６文字の文字列）をテーブル・サイズ ℓ_0 で

〔図5－14〕ディクショナリー・テーブル



割り余りmを求める。

②，テーブルの先頭からm変位した場所を調べもし，空セルがあればそこを登録位置とする。

③，もし使用中のセルであればさらにmに定数Kを加えm＝m＋Kとして②の操作にもどる。この時m＞l_0になったらm＝m－l_0の操作を行う。

TDT部はバーチャルな部分も含めてその大きさはLである。

TDTにアイテムをマッピングする手順は次の通りである。

①，テーブルサイズをLと考えて，アイテム名標をLで割り余りを求める。

②，この結果アクセスすべきページn_0とそのページ内相対アドレスm′が決定する。

③，n_0を実際に主記憶にロードし，その相対アドレスm′の位置を調べる。もし空セルであればそこに登録し処理を終える。

④，もし使用セルであれば一定長K′を加えm′＝m′＋K′として③の後半の操作にもどる。この時m′＞l_1になったらm′＝m′－l_1の操作を加える。

アイテムのサーチに関しても，今述べたことと全く同様にして行われるが，目的とするアイテムがRDT部に格納されているかTDT部に格納されてい

るかはＲＤＴ部をサーチしてみなければ判別できない。即ちＲＤＴ部をサーチしその間にコンフリクト回数がある一定値を越えても目的とするセルが見つからない時は改めてＴＤＴ部にアクセスを開始するという形式をとる。

以上外部名標（アイテム名標，セット名標）でディクショナリ・テーブルにアクセスする場合の処理アルゴリズムについて説明したが，内部名標（インターナル・ネーム）からアクセスすることが必要な場合もある。

内部名標はそのセルのテーブル相対アドレス値が採用されているから，内部名標によるテーブルのダイレクト・アクセスが可能である。

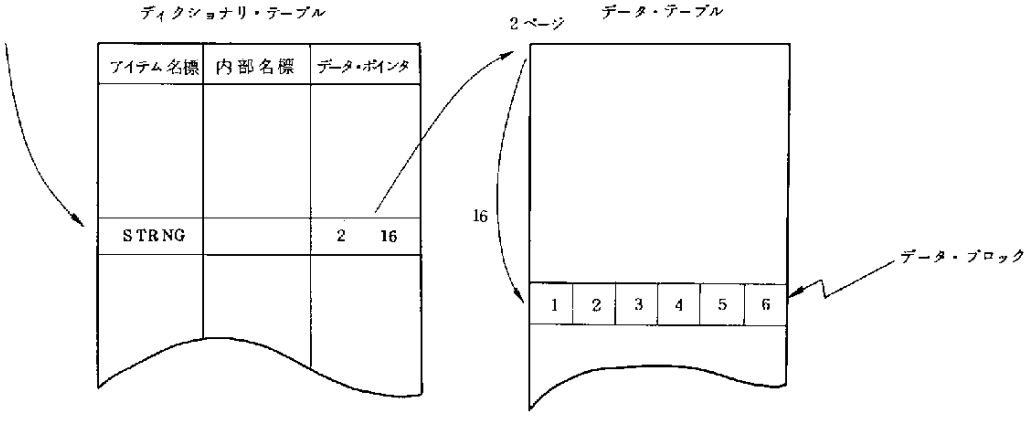
５．５．２　データ・テーブルのアクセス

アイテムのデータ部は，宣言時にスタティックにアロケートされる，アロケーションの方法については，データ・ページの管理の項で述べた通りである。ここでは，実際のデータの格納検索がいかにして行われるかについて説明する。

ａ，データの格納

データの格納要求は，対象とするアイテム名標と，データブロックのセル番号を指定することによって出される。

〔図５－１５〕データ・テーブルのアクセス機構



例えば"STRNG"という名標のアイテムの３番目のセルにデータをストアすることを考えると，これは次の様な過程で実施される。

①，ディクショナリ・テーブルをアクセスし"STRNG"の格納されている位置を知る。

②，その位置のデータ・ポインタを調べることによりアクセスすべきデータ・ページとそのページ内に格納されているデータ・ブロックのページ内相対アドレスを知る。

③，データ・ブロックセル番号3の位置に目的とするデータを格納して処理を終る。

データの検索も全く同様な操作で目的とするデータを得ることができる。

5. 5. 3 セット・テーブルのアクセス法

A，セットの登録

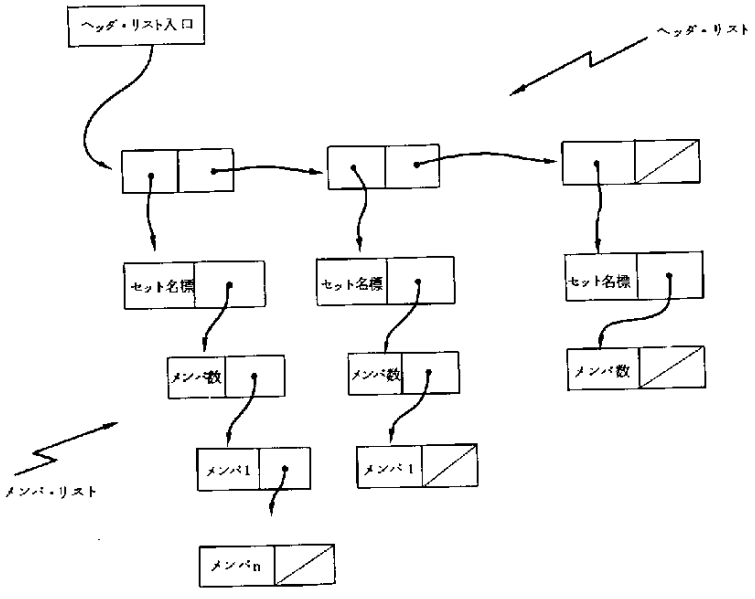
セットに対するページの割り当ては宣言時にスタティックに行われる。この過程は次の様に要約される。

① ディクショナリ・テーブルにセット名標を登録し対応する内部名標を決定する。

② セット・ページから割り当てページを1つ確保し，そのページ番号をディクショナリ・テーブルに記録する。

③ ページ・ヘッダのセット・ヘッダ・リストの入口を調べ，セット・ヘッダ・リストをたどる。

〔図5－16〕 セット・テーブルのアクセス機構



④，ヘッダ・リストのテイルに達したらフリーリストよりセルを確保しそのテイルに新たにセット名標の追加を行う。

この様にセット・ヘッダ・リストはそのページに格納されているセット名標をリストしたものである。

B，セットのメンバの追加

あるセットのメンバが定義されると，メンバはセット・テーブル内のそのセットのメンバ・リストにつながれる。メンバ・リストに到達する過程は上で述べた手順と全く同じである。

（メンバはアイテムの内部名標である。）

メンバ・リストは内部処理の都合上内部名標に関して順序化されたリストとして作成される。

メンバの除去も又同様な過程を経て実施される。

C，セット演算に関して

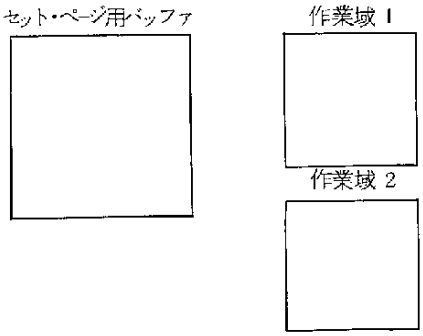
セット・ページのアクセスに関連して，PUT文，FUNCTION文等で扱われるセット・エクスプレッションの評価法について簡単に説明を加えておく。

セット・エクスプレッションはプリコンパイル時に評価され次の形式のセット演算式に変換される。

セット名標1 | セット演算子1 | セット名標2 | セット演算子2 | ……

これは内部的に次の様に処理され目的とするセットを得る。

〔図5－17〕 セット演算用領域



セット演算を行うための作業域を2つ設けセット用バッファーとどちらからの作業域，例えば作業域1に演算対象となるセットを読み込む。

演算結果は残りの作業域２に作成する。

セット・ページ用バッファに次のセットを読み込み作業域２の結果と演算し結果は作業域１に作成する。以下同様に作業域を交互に使って演算が終了するまで続ける。

この様にセット演算は，常に２つのセットの間で行われる。セットのメンバーは内部名称で順序化されているからその演算はかなり効率的に行うことができる。

演算のアルゴリズムは一般によく知られた方法をとっているが，ここでは和集合を例に一例を説明しておく。

(1) 和集合

２つのセットＬ，Ｍの和集合は次の方法で求まる。

Ｌ：（ℓ_1，ℓ_2，ℓ_3，ℓ_4 ……ℓ_I ）

Ｍ：（m_1，m_2，m_3，m_4 ……m_J ）

ここでセットの要素は小さい順に並べられているものとする。

セットＬ，Ｍの現在比較されている要素の位置を示すものをリーダと呼び各々ｉ，ｊとする。

step 1：リーダを初期化しｉ＝１，ｊ＝１とする。

step 2：ℓ ｉ＝ｍｊならば

１ｉが求める要素となり，ｉ＝ｉ＋１，ｊ＝ｊ＋１とする。

ℓ ｉ＞ｍｊならば

ｍｊが求める要素となり，ｊ＝ｊ＋１とする。

ℓ ｉ＜ｍｊならば

ℓ ｉが求める要素となり，ｉ＝ｉ＋１とする。

step 3：ｉ＞Ｉならば

m_{j+1}～ｍＪまでが求める要素となり処理は終了

あるいは

ｊ＞Ｊならば

ℓ_{i+1}～ℓ_Iまでが求める要素となり処理は終了

その他の場合はｓｔｅｐ２へもどる。

５．５．４　連想テーブルのアクセス法

Ａ，アクセス・ワードのページ割り当てについて

問題点の項でも述べた様に，各スペースにおけるアクセス・ワードに関して，それに関するトリプルの定義はアクセス・ワード毎に偏りを生じる。このため各ページに収容するアクセス・ワードの個数を一定として機械的に扱うとあるページのトリプルが急激に増加しページ・オーバフローを起す可能性がある。

このため次のページ割り付けコントロール・テーブルを設けプログラムの進行に伴うトリプルの生成過程を監視し，新たにアクセス・ワードにページを割り付ける必要が生じた時はこのテーブルを参照して割り当てを行う様にする。

〔図５－１８〕　ページ割り付けコントロール・テーブル

ページ番号	P	P＋1	P＋2	P＋3
A－SPACE	n_{11}	n_{12}	n_{13}	n_{14}
O－SPACE	n_{21}	n_{22}	n_{23}	n_{24}
V－	n_{31}	n_{32}	n_{33}	n_{34}

〔図５－１８〕に示す様に順次に編成される４ページを単位としてテーブルに登録し，これ等のページをアロケーションの対象とする。

各スペースに関してトリプルが定義されるとその対象するページのトリプル個数をカウント・アップし，テーブル内に置かれたページの利用状況が分る様になっている。

新たにアクセス・ワードを割りつける必要が生じた時は，その利用状況を調べて最も使用頻度の低いページに割り当てを行うようにする。又，あるページに関して，その収容トリプル個数が一定値を越えたら，４ページすべて

をテーブルから取り除き新たな空ページをテーブルに登録する。テーブルの管理からはずされたページに関しては以後アクセス・ワードの割り当ては行われない。

B，トリプルの創成に関して

トリプルが連想テーブルに創成される過程は次の様に要約される。

①，指定されたトリプルの各要素をディクショナリー・テーブルから内部名標に変換する。

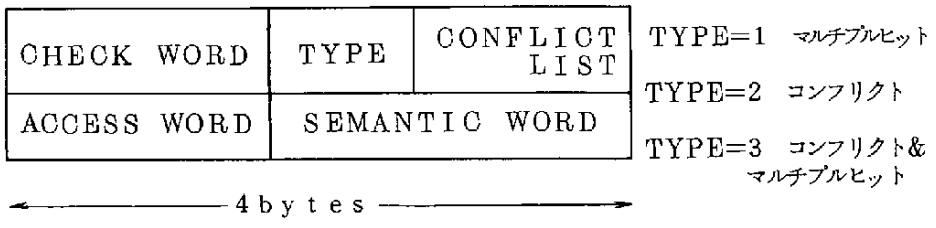
②，トリプルの第1項の内部名標からトリプルを格納すべきページを知り，もし主記憶中にそのページがなければスワップインする。

③，トリプルの第1項，第2項をダブル・ハッシングすることによりページ内格納アドレスを知る。

ダブル・ハッシングの方法として各種の方法が考えられるが，ここでは内部名標同志のExclusive Orをとっている。

④，ここで，到達したセルの状態により，その後の処理は3通りの場合に分れる。〔図5－19〕参照

〔図5－19〕 TRIPLEの基本形



④－1，ノーマルな場合

発見したセルのチェック・ワードが0の場合で，この場合はそのセルが未使用セルであることを示すから，その位置にトリプルの登録を行えばよい。

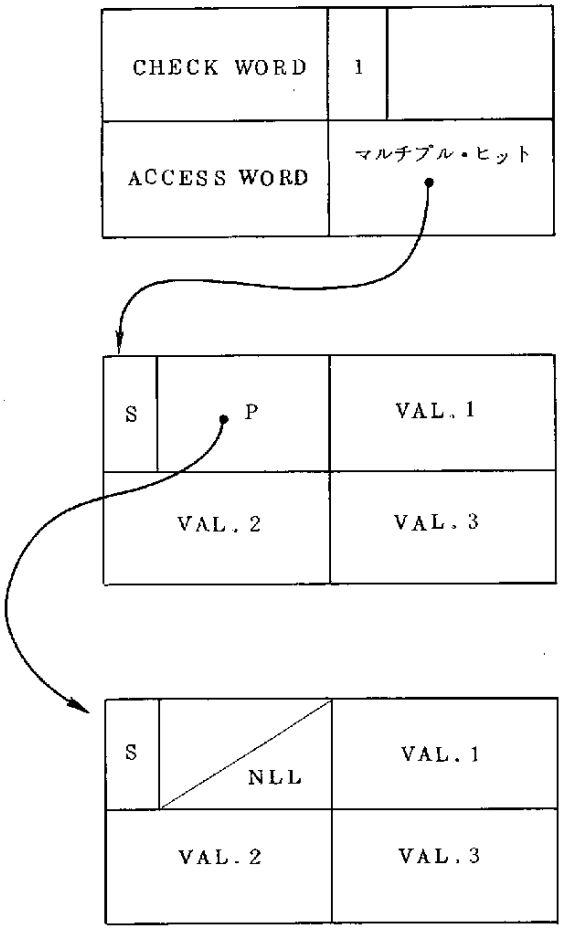
④－2，マルチプルヒットの場合

発見したセルのチェック・ワードが0でない時はそれぞれチェック・ワ

－ド，アクセス・ワード同志の比較を行う。もし両者が共に一致すればマルチプルヒットであることになる。

この場合〔図５－２０〕の様にそのページのマルチプルヒット，コンフリクト用領域から空セルを確保しそのセルのセマンティック・ワードを空セルに移し，セマンティック・ワード部はその空セルに対するポインタにおきかえる。さらに対象とするトリプルのセマンティック・ワードを格納する又セルのＴＹＰＥ部にはマルチプルヒットの表示を与える。もし，すでにマルチプルヒット・リストがつけられている時にはそのリストをたぐり終端のセルに対象とするトリプルのセマンティック・ワードを登録する。

〔図５－２０〕　マルチプル・ヒットの処理

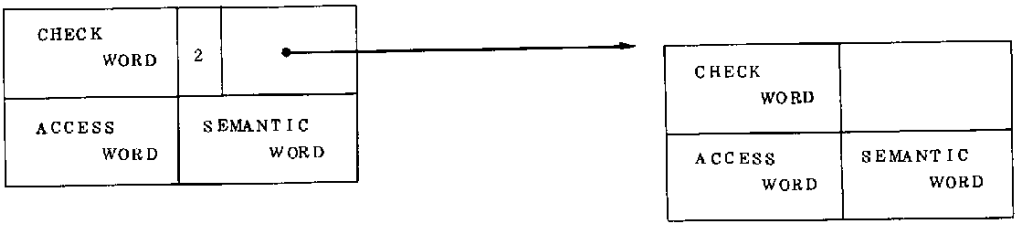


④－３，コンフリクトの場合

発見したセルのチェック・ワードが０でなくかつマルチプル・ヒットでもない時はコンフリクトにあたる。

この場合〔図５－２１〕に示すように，マルチプルヒット，コンフリクト用領域から空セルを確保し，コンフリクト・セルのコンフリクト・リストからこの空セルに対してポインタをつける。対象とするトリプルは空セルに格納される。又ＴＹＰＥ部には適当な表示を行う。

〔図５－２１〕　コンフリクト・セルの扱い



もしコンフリクト・リストがすでに作成済みの場合は，そのリストをたどり空セルをその終端につなぎトリプルの登録を行う。

以上トリプルの生成に関して，Ａ－Ｏスペースに限って話をしたが，Ｏ－Ｖスペース，Ｖ－Ａスペースについても全く同様な操作を行わなくてはならない。

Ｃ，トリプルの検索に関して

トリプルの検索に関して，トリプルの２項が与えられて検索する場合，即ちｓａ′ｆ表現のＸ・Ｙ＝？，Ｘ・？＝Ｚ，？・Ｙ＝Ｚの場合に対してはトリプルの創成とほぼ同じ過程をたどることにより目的とするアイテム集合を得ることができる。

一方，トリプルの１項が与えられて残りの２項を求める質問形式（Ｓａ′ｆ表現ではＸ(?)＝？，？(?)＝Ｚ，？(Ｙ)＝？に相当する）に対しては，対応するアクセス・ページをすべてシーケンシャル・サーチしなくてはならない。

5．6　コマンド・リストについて

トリプルの創成，除去，データの格納，セットへのメンバの追加，除去等を行うためには実際に対象とするページを主記憶にロードしそのページに対して書き込みを行う必要がある。

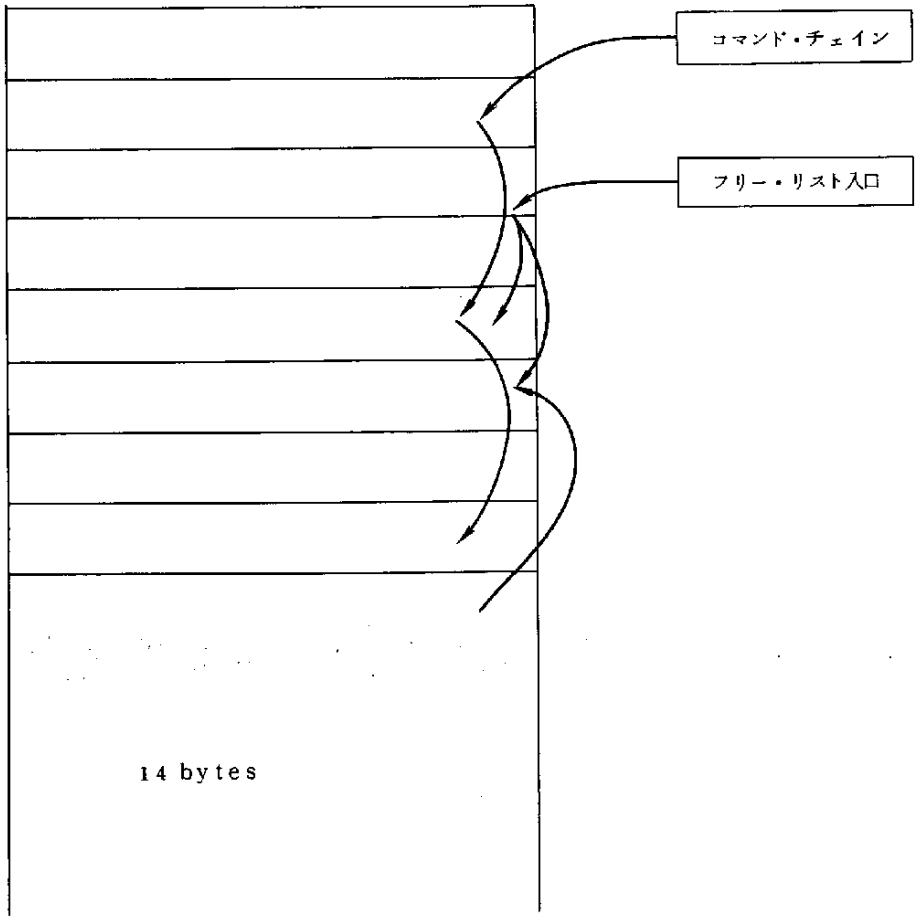
この時，上記の書き込み指示に関するステートメントが実行されるたびに対象とするページをロードするためにスワップ・アウト，スワップ・インを行ったのではページ・トラフィックが余りに頻繁すぎて効率が著しく低下してしまう。

この様な事態を避けるために，バッファとしての役割を果すコマンド・リストを設け効率の改善を計っている。

コマンド・リストは主記憶中に置かれた連続領域で，〔図５－２２〕に示される様に１４ｂｙｔｅｓを１単位セルとして構成され，書き込みコマンドを蓄えたリストとして管理される。

即ち書き込みに関するステートメント，例えばトリプル創成ステートメント（ＭＡＫＥ文）等が実行されると，直接連想テーブルへの書き込みは行われずに一担コマンド・リストに登録される。

〔図5－22〕 コマンド・リスト

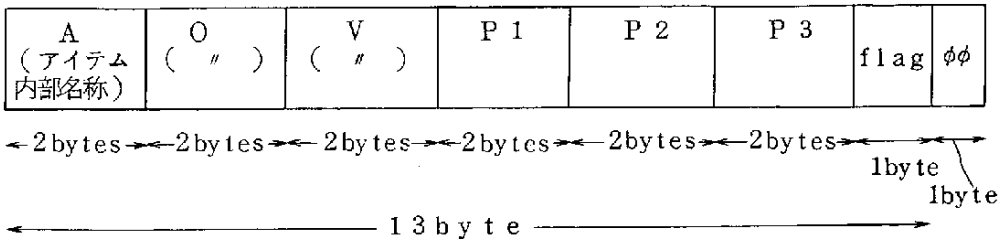


この時書き込みに必要な情報は以下に示す様な形式のコマンドに変換されフリー・リストから空セルを一つ確保しそこに登録される。

◦ コマンドの形式

ⅰ）トリプル（A_1，O，V）に関するコマンドは次のような形式をとる。

〔図５－２３〕



P１：Aの連想テーブルにおけるアクセスページナンバー

P２：O　　〃

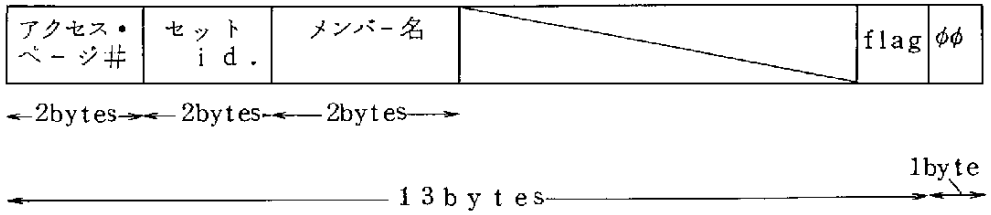
P３：V　　〃

flag：X　Fφ　の時トリプルの創成を指示する

　　：X^0　F１　〃　トリプルの消去を指示する

ⅱ）セットに関するコマンドは次のような形式をとる。

〔図５－２４〕



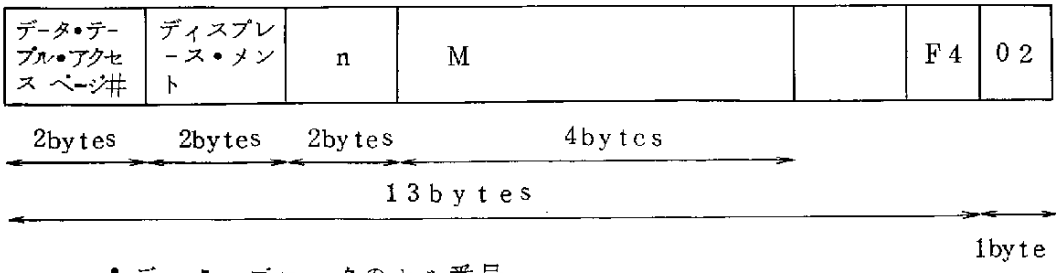
flag　：メンバーの追加

　　：　〃　削除

　　：全メンバの除去

iii）データに関するコマンドは次のような形式をとる。

〔図５－２５〕



n：データ・ブロックのセル番号

M：格納すべきデータ

実際の書き込みが行われるのは次の様な場合である。

①，コマンド・リストのフリー・リストを使い果しコマンド・リストがオーバーフローの状態におちいった時。

この時は任意のページ番号を１つ選び，そのページ番号のページを主記憶にスワップインするためのＩ／Ｏ動作を起す。

Ｉ／Ｏ動作と並行してコマンド・リストをサーチしそのページ　番号を有するコマンドをことごとくチェインしその入口をコマンド・チエイン入口として記録しておく。

Ｉ／Ｏ動作の終了を待ってチェインされたコマンドを対象とするページに書き込む。書き込みが終了したらそのコマンド・チェインは必要がなくなるからそのままフリー・リストに返却する。

②，検索命令等の実施によりあるページが新たに主記憶に読み込まれた時。

この時も同様にそのページ　番号を有するコマンドをチェインし読み込まれたページに実際に書き込みを行う。

③，現在主記憶にあるページをスワップ・アウトする必要がおきた時。

同様にそのページ番号を有するコマンドをことごとくそのページに書き込んでからページのスワップ・アウトを行う。

あとがき

コンピュータ・グラフィックスにおいて最もよく問題とされるのはコストの問題である。これには①装置の絶対コスト，②ソフトウェアの開発コスト，③コスト・エフェクティブネスの3点が含まれる。

装置コストに関してはグラフィックスの普及による量産体制からコスト・ダウンを計ること又低価格装置の開発を計ること等が考えられるが，これらはいずれもハードウェア自体にかゝわる問題が大きい。

ソフトウェアの開発コストに関しては，そのソフトウェアの体系化を進めアプリケーション・システムの開発を容易にするような環境を作ることが大切である。

又コスト・エフェクティブネスの問題に関しては，利用形態の改善を図ること（例えばホスト・サテライト系で利用する等），又利用コストに見合うだけの成果をあげるような効果的なアプリケーション・システムを多数開発することが必要である。これにはプログラムの専門家に限らず多くの応用分野の人々（物理，化学，建築，機械……etc）にシステムを開放しその参加を要請する必要がある。この様にプログラムの非専門家に対しても，その利用を容易にする様な適当なソフトウェアの開発が必要になってくる。

以上述べた事柄に解答を与える一つの方向としてハイレベルな汎用グラフィック言語の開発が考えられる。UNGLはこうした目的から特に汎用性に重点を置いて作成されたグラフィック言語である。その汎用化のポイントとして言語形成はプロセデュラルなものとし，又従来のコンパイラ言語をホスト言語とするベース言語アプローチをとった。機能的には図形データ処理，データ・ストラクチャ操作，割り込み処理に関する扱いを考慮し，又マン・マシン系に於ける図形情報を介した問題解析を行う上での便宜も考えた。これは内部的に図形構造と問題構造を分離して扱うマルチ・レベル・データ・ストラクチャとして実現されており，両者は内部的に対応関係がとられている。しかし2つのデータ構造は独立に管理することが可能であり，モデル表現にフレキシビリティを持た

せている。

しかし，最近ではこうしたすべてのアプリケーション・プログラムの実現を可能にするような言語を指向すること自体が無理であるとし疑問視されている向きもある。（例えば1972年バンクーバにおけるIFIP Working Conference on Graphic Language でのSession 7 panel discussion "Are we anywhere near universal graphic language"参照）即ち一つはあらゆるアプリケーションから要求される各々異った機能，特性を一つの言語に反映することは実現不可能であろうということであり，又もう一つは仮にその様な言語が実現されたとしても，汎用性と効率の相反する性格からその効率が著しく損われる恐れがあるという点である。

UNGLもこうした点から見れば必ずしも完成された汎用言語とはいえない。しかしコンピュータ・グラフィックスの本質である「マン・マシン・コミュニケーションによる問題解析」を一つの中心的な課題と考え，その様な性格を備えたアプリケーション・プログラムの実現に対し有力な手段となることは確かであろう。これは必ずしも完成されたアプリケーション・プログラムの開発にはつながらないかもしれないが，グラフィックスの持つ可能性をより広範囲の層に認識させ，そこに新たな発想を生み出し真に効果的なグラフィックスの利用分野を切り開き，あるいはその限界をはっきりさせていくことにつながるであろう。

この様に，UNGLは数少ないグラフィック言語の中の一つとしてまだ試作的な域にある。先に述べた効率の問題，各機能のバランスの問題等については未だ検討の余地を多く残している。これは言語の使用経験，今後の応用分野の動き，ハードウェアそのものの変遷等に注目しながらリファインしていかなくてはならないだろう。

参 考 文 献

1. An experimental display programming language for the PDP-10 computer
   by William Newman
   October, 1969
2. GRAF: Graphic Additions to FORTRAN
   by A. Hurwitz and J. P. Citron
   SJCC, 1967
3. GPL/I extension for computer graphics
   by Dauid N. Smith
   SJCC, 1971
4. Display Procedures
   by William M. Newman
   ACM October, 1971 vol 14 №10
5. A system for interactive graphical programming
   by William M. Newman
   SJCC, 1968
6. Principles of interactive systems
   by C. I. Jonson
   IBM System Journal vol 7 №3, 4 1969
7. The LEAP language and data structure
   by Paul D. Rovner and Jerome A. Feldman
8. An Algol-based associative language
   by Paul D. Rouner and Jerome A. Feldman
   ACM August, 1968 vol 12 №8
9. Auxiliary-storage associative data structure for PL/I
   by A. J. Symond
   IBM System Journal №3, 4 1968

10. The list set generator : A construct for evaluating set experessions

by Stuart C. Shapiro

ACM December, 1970 vol 13 №. 12

11. ディスプレイシステムの開発（45-S002）

（財）日本情報処理開発センター

12. 汎用図形処理言語の開発（46-S002）

（財）日本情報処理開発センター

昭和４８年３月発行

発行所　財団法人　日本情報処理開発センター
東京都港区芝公園３－５－８
機械振興会館内
TEL（434）8211（代表）

印刷所　山陽株式会社
東京都港区芝琴平町１９番地
TEL（591）0248

47-S005