OUKA
The University of Osaka
Institutional Knowledge Archive

Title	計算機言語処理システムの構成に関する研究
Author(s)	首藤, 勝
Citation	大阪大学, 1973, 博士論文
Version Type	VoR
URL	https://hdl.handle.net/11094/2654
rights	
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

The University of Osaka

計算機言語処理システムの構成に関する研究

1973年1月

首藤　勝

内　容　梗　概

本論文は，著者が三菱電機株式会社において電子計算機の方式およびソフトウエアの研究にたずさわつてきたなかで，計算機言語処理システムの構成に関する一連の研究成果をまとめたもので，全体を3編で構成している。各編第1章は緒論としてその研究の目的および必要性について述べ，また各編最終章ではそこで得た結果を示し結論とした。

第Ⅰ編はマイクロプログラミング技術を活用したコンパイラの構成を対象とし，特にマイクロプログラムを随時交換可能な計算機での効果的なシステム構成法とその実例を示した。

第2章ではマイクロプログラミングによるオブジエクト計算機の構成法を述べた。交換可能なマイクロプログラムによつて計算機の構造とソース言語の構造との隔りを軽減し，コンパイラの構成を単純化できることを示している。

第3章ではコンパイラの機能のうち重要な表管理に統一的手法を導入しコンパイラ処理容量の制限を緩和する方法を述べ，マイクロプログラムによりこの表管理の基本機能を実現してコンパイラの処理効率を高める方法を示した。

第Ⅱ編はコンパイラ作成過程への機械的手法導入に関し著者の得た結果をまとめたものである。

第2章では，コンパイラの原形を手書きで作成し，機種ごとに異なる部分を置き換えて所要のコンパイラを得る手順を考え，その過程を計算機に実行させることにより機械的にコンパイラを作成する方法を提示した。実際に2種の計算機を用いてこれを行ない，得られたコンパイラを使用してそのオブジエクトプログラムの効率低下による実行時間の増大が約1.3倍に止まることを示している。

第Ⅲ編では，特に制御用計算機応用に於て効果的に適用し得る言語システムの構成の問題をとり上げた。

第2章では，制御用計算機の応用に於て従来の標準言語で不足する機能を言語拡張により実現する方法とその実施例を述べている。

第3章ではコンパイラ言語よりさらに制御対象と密着した高位言語を示し，その翻訳と実行のためのプログラムの構成について述べている。これはプログラミングの省力化に効果を持つ上に，プラントの現地調整に即応したプログラム変更を現地計算機でソース言語レベルで行なうことを可能にしたものである。

第4章ではオンライン計算機におけるプログラム検査効率向上に有効なシミュレーション法について，そこに生じる問題点とその解決法を実例によつて示した。

最後に，以上の3編の研究結果を要約して結言とした。

Summary

This thesis, consisting of three parts, summarizes the results of the study on the organization of language processing systems by the author during his research and development activities at Mitsubishi Electric Corporation. The first chapter of each part is the introduction to the respective part and describes the objective and the necessity of the study, and the last chapter of each part is the conclusions of the respective part.

Part I treats the compiler organizations using microprogramming techniques, especially in the systems in which the microprograms can be exchanged through program control.

Chapter 2 describes a method of organizing a object computer with microprogramming. With the exchangeable microprograms it is possible to reduce the difference between the structures of the source language and the machine and to simplify the organization of the compiler.

Chapter 3 describes a method of dynamic table control in which a unified dynamic storage allocation scheme is applied to the various tables and stacks in a compiler. This method can reduce the limitations of processing capacities of the compiling system. Each elementary function of the dynamic table control is realized with an exchangeable microprogram. Some results show an significant increase of efficiency of the compiler.

Part II is a result of the study on automatic compiler generation obtained by the author.

In Chapter 2 is proposed a method of automatic compiler generation. An original model of compiler is first written by hand coding. Some parts of this compiler are replaced for another machine, and a series of processes are executed mechanically to yield the final result. This method was applied to two existing computers, and some object programs were obtained through the mechanically generated compiler. An 1.3 times of increase is reported in the execution time of those object programs.

Part III treats the programming languages and the system organizations which are effective in the process control applications.

Chapter II describes an extended language for process control in which those functions required in control applications are added to a conventional standardized programming language.

In Chapter 3 is proposed a high level problem-oriented-language in which every feature has a closer correspondence to some control algorithm in comparison with the case of the compiler languages. This type of languages is powerful in the reduction of programming labour and in the preparation of program documentation. There is also described a proposed system

organization for both the program composition and the program execution. In this system some features are adopted to enable the tuning and the correction of programs in the level of the source language at the site where the plant is installed.

Chapter 4 describes a method of on-line simulation effectively applicable to the program debugging in process control computers, referring to the problems arrising in the actual implementation and to some solutions to them.

The Conclusion Part summarizes the results of the whole three parts.

計算機言語処理システムの構成に関する研究

目　　次

緒　　論

電子計算機を構成する回路素子および実装の技術は近年著しい進歩を示し，ハードウエア面の性能向上は顕著なものがある。また計算機の利用面でも，計算機で解くべき問題の規模の増大と利用法の多様化によつて，ますます高度なあるいは複雑な利用形態が計算機に課せられるに至つている。計算機のハードウエアと利用者の間に介在するソフトウエアの構成と動作効率が全システムの機能発揮に及ぼす影響は極めて大きい。この傾向は今後も変らないと考えられソフトウエアの構成技術が当分の間重大な問題となるであろう。

ソフトウエアを大別すると，ハードウエアの制御とプログラムの基本的な動作の管理を主目的とする“制御管理プログラム”（オペレーティング・システム），各分野の応用問題を記述するために使われる“プログラミング言語”とそれをハードウエアで実行可能な言語に翻訳する“コンパイラ”，および各種のプログラミング言語を用いて書かれる“応用プログラム”（個々の問題を解くためのプログラム）の三つになる。このうちのプログラミング言語とコンパイラの分野について，実用上重要となるのは次の諸点である。

(1)　プログラミング言語の構成の問題

(2)　コンパイラの構成の問題

(3)　コンパイラ開発過程の効率化の問題

これらは相互に関連をもちつゝ研究開発の対象となつており，その成果がまたソフトウエアの他の分野，さらに計算機そのものの構成技術の上に波及しつゝある。

本論文は著者が三菱電機㈱において数年間にわたり実施した研究とその実用化の成果をまとめたもので，全体を3編で構成している。この3編は具体的な問題のとり上げ方はそれぞれ異なるが，計算機言語処理システムの構成法を追求してより高能率のシステムを実現するという目的でなされた研究の報告であり，本質的に同じ問題を取扱つているといえる。

第Ⅰ編ではマイクロプログラミング技術を活用してコンパイラの構成をより効率よくする方法を述べる。ここではマイクロプログラムを随時交換することが可能な計算機を採り，マイクロプログラミングをソフトウエアの技術として活用する。この場合のコンパイラの構成には2面からのアプローチが有効であり，実行時の基本機能をマイクロプログラムで実現する問題を第2章で，コンパイラそのものの基本機能をマイクロプログラムで実現する問題を第3章でそれぞれ扱かう。

第2章で述べるマイクロプログラミングによるオブジエクト計算機の構成は，交換可能なマイクロプログラムを用いることにより計算機ハードウエアの論理構造とコンパイラのソース言語の構造との隔りを軽減し得る点が特徴的である。これによりコンパイラの構成を単純化できるため，同一規模の計算機でより高度な仕様をもつソース言語の翻訳処理が可能となる。COBOL コンパイラの実例でこれを示す。また，COBOL のような10進可変桁処理を基礎とする言語に対して作られたオブジエクト計算機構成用のマイクロプログラムは，同系統のデータ構造と対象とする事務用問題向き言語ＡＣＥ（RPG型言語）に対してもほゞそのまゝ使用することが可能であることにもふれる。

第3章では，コンパイラで多用される表およびスタックのメモリ割付けの制御を統一的に管理し，表領域を効果的に使用してコンパイラの処理容量上の制限を緩和する方法をとり上げる。この方法自体はマイクロプログラミングとは関係なく使用できるが，表管理のための基本的な諸機能をマイクロプログラムで実現することにより，コンパイラ記述用言語機能としてそれらを用いることができるため，コンパイラの構造を単純化し，とくに保守性の秀れたシステムを実現することが可能となる。ここでも交換可能なマイクロプログラムが活用される。

第Ⅱ編はコンパイラ作成過程への機械的手法導入に関し著者の得た結果をまとめたものである。この問題についてはかなり多くの研究が報告されているが，ここでは新機種の開発に際して早期にコンパイラを使用可能とするための転作を中心に考える。コンパイラの原形を手書きで作成し，機種ごとに異なる部分を置きかえて所要のコンパイラを得る手順を考え，その過程を計算機処理に適

した形で構成して機械的に他機種用コンパイラを作成する方法を提示する。

第Ⅲ編では，制御用計算機応用に適用するための言語システムを採り上げる。制御応用では一般用と比べてオンライン性が高度であること，特殊な装置が結合されていること等の条件から，計算機処理したがつてプログラミング言語に対しても異なつた要求が提起される。従来この要求に充分応え得る高位言語システムが供給されなかつたために，アセンブラ言語が現在なお多くの部分で使用されるといつた事情にある。制御用プログラミング言語にも2面のアプローチがある。既存のコンパイラ言語に対し言語機能の拡張を施して制御用言語を形成する方法と，コンパイラ言語よりさらに制御対象と密着した高位言語，いわゆる問題向き言語を用いる方法である。ここではこの両者について，必要な機能，その実現法を述べ，実例によつてそこにおける問題を明らかにする。また，制御用計算機は最終的にプラント現地に設置されるものであり，その時のシステム構成は必ずしもコンパイル実施，プログラム修正等プログラム作成作業に適するものではない。このため，現地での作業のために対策を講じる必要がある。この点についても実例をもつて述べる。

関連発表論文

(1) 首藤，関本，魚田，鶴岡：“MELCOM−1530 COBOL システムの構成”，昭39信学全大，554（昭39）

(2) 首藤，関本，鶴岡：“MELCOM−1530 COBOL システムに於ける言語変換機構とオブジエクトの構成”，昭40電四連大，638（昭40）

(3) 国分，有坂，首藤，魚田：“MELCOM−3100ソフトウエア(5) − ACE コンパイラシステムの概要 − ”，三菱電機技報，42−10，P.1360（昭43−10）

(4) 首藤，小碇：“動的な表管理とCOBOLコンパイラへのその応用”，情報処理学会誌，13−2，P.113（昭47−02）

(5) 首藤，魚田，居原田：“ALGOLコンパイラ向き基本言語試案”，昭39信学全大，553（昭39）

(6) SUDO，UOTA，IHARADA：“Some Considerations and Experiments on the Basic Programming Language”，MITSUBISHI DENKI LABORATORY REPORTS，7－1，p.40（Jan. 1966）

(7) 首藤，魚田，居原田：“計算機を用いたコンパイラ作成自動化の実験”，情報処理学会第7回プログラミング。シンポジウム報告集，C－53（昭41－01）

(8) 首藤，魚田：“コンパイラ自動作成の一方法”，信学誌，50－4，P.649（昭42－04）

(9) 有田，井上，首藤，居原田：“MELCOM－350／30オンライン。シミュレータ”，三菱電機技報，44－5，p.644（昭45－05）

(10) 有田，首藤・関本，居原田：“プロセス制御用プログラミング・システムの構成”，昭45信学全大，941（昭45）

(11) 有田，首藤，関本，居原田：“プロセス制御用基本コンパイラ言語”，昭45信学全大，942（昭45）

(12) 有田，首藤，関本，居原田：“プロセス制御用コンパイラCONFORM”，三菱電機技報，45－2，p.213（昭46－02）

(13) 首藤，五十嵐，居原田：“プロセス制御用プログラミング・システム－MDSS－”，昭47信学全大，1196（昭47－04）

(14) 居原田，五十嵐，首藤：“ソース言語レベルでのオンラインプログラムのチューニングについて”，情処学会第13回大会，31（昭47－12）

(15) 中原，五十嵐，居原田，首藤：“プロセス。データ・モニタリングにおけるファイルの一構成”，情処学会第13回大会，126（昭47－12）

第Ⅰ編　マイクロプログラミングによるコンパイラの構成

第１章　緒　　論

マイクロプログラミング技術はWilkes[1]により提案されて以来，多くの影響を計算機の設計技術の上に与えて来た。マイクロプログラミング方式によつて，計算機の論理構造を金物として固定せず，系統的な制御が可能となるため，融通性が増し，設計法・試験法が合理的になる。この技術は読出し専用メモリ（ROM）の発達とともに実用化され，現在の主流をなす機種に採用されることによつて，その立場を確固たるものとした感がある。Wilkesはマイクロプログラミング技術をハードウエアの技術とみており，制御メモリを動的なものにする，いわゆるWritable Control Memory（WCM）に関しては消極的であつたようだ。その後の技術の発達と応用面からの要請によつて，このWCMに対する認識が高まりつゝあり，特定応用に対する性能／コスト比向上，設計変更への適合性などに活かそうとする考え方が出て来ている。[9]

マイクロプログラミングの適用は従来殆どハードウエア設計の面に向けられており，ソフトウエアの構成，性能，融通性の改善に利用する動きはFORTRAN[5][6]のプログラムをコンパイル手続きを省略して，実行させようとするものを除いてあまりない。これはこの技術がROMと密接に関連しつゝ発達したことによるものであろう。

現在の計算機では極めて複雑な構成をもつオペレーテイング・システム，コンパイラなどのシステムプログラムを備えることが普通になつており，これらが基本ソフトウエアとしてハードウエアを包んだ形で存在し，ユーザが問題を解くために書くプログラムは，その外側にあるもの（応用ソフトウエア）と位置づけされている。基本ソフトウエアの多くの部分は，ある範囲の適用面については必須のものであり，計算機の構成要素の一部と見ることが適当な場合も多い。従来はハードウエアコストが大きかつたことと，ソフトウエアの技術がまだ成長期でシステムの形が流動的であつたことのために，これらの部分はソ

フトウエアとして作られることが適当であつたが，高密度半導体を始めとするハードウエアの発達とコストの低下，ソフトウエアの機能モジユールの考え方とその技術の進歩とによつて，従来ソフトウエアの形で作られたこれらの機能部分をハードウエア化して行く可能性が強くなつている。これらは元来プログラム処理されるに適した性質を持つ部分であるから，ハードウエア化は，マイクロプログラミングを前提とした形で行なわれるとみるべきであろう。

ここでは，マイクロプログラミング技術をソフトウエアの構成に適用する問題を，基本ソフトウエアの中で重要なものの一つであるコンパイラとその実行時システムをとり上げて論じる。コンパイラは，ソース言語のプログラムを機械語に変換する機能をもつものであるが，機械の論理構造と，ソース言語の文法構造との隔りの如何がコンパイラの構成の難易に直接影響する。そこで両者の中間的な実行時系を構成し，それをマイクロプログラミングにより実現する問題と，そのレベルまでの翻訳を受持つコンパイラ自体を構成する基本的な機能要素を，マイクロプログラム化する問題の二面から，この課題を捉えることにする。実行時系の例としてCOBOL 用のオブジエクト系を採り上げるが，これは2進処理を中心として作られた計算機と，大きく相異する文法をもつ言語への適用として，この構成法の評価が問われるものである。

第2章　マイクロプログラミングによるオブジエクト計算機の構成

2.1　緒　　言

マイクロプログラミング技術の適用によつて計算機の構成を簡略にしてその機能面の融通性を増大することが可能であるが，通常は計算機の機械語レベルよりも下層（回路に近い側）にマイクロ命令語を設定し，マイクロプログラムを特別な記憶装置に貯える形式をとつている。このため，マイクロプログラミング技術はハードウエアの構成技術としてのみ利用されており，機械語レベルより上層のプログラミング言語系の構成についてはマイクロプログラミングによらない従来の計算機と変るところがない。ここでは，マイクロプログラムをも主記憶装置に貯え，プログラムによる制御を用いてマイクロプログラムを随時交換することを可能とした計算機をとり上げる。この形式の計算機では，使用目的に応じて計算機の基本命令系を変更できるため機能上の融通性が極めて大きくなる。

この形の計算機では，コンパイラのオブジエクトプログラムを構成する言語は固定された機械語系ではなくマイクロプログラムで組み立てられた実行時命令系となる（ここではこれをオブジエクト計算機と呼ぶ）。このオブジエクト計算機をソースプログラム言語の性質に応じた形に構成することによつて，ハードウエアのもつ語構成および基本演算機能とは独立にソース言語文法が要求する機能をもたせ，これがソース言語と機械構造の隔りを減らせることになつて，効率よいコンパイル処理を可能とすることができる。

2.2　ストアドロジツク形マイクロプログラミング方式

ここではマイクロプログラムを主記憶装置に貯えてそれによりハードウエアのもつ処理単位より大きい機能単位を基本命令として扱える計算機をとりあげる。この形のマイクロプログラミング方式をストアドロジツク形と呼んでいる。この形式の計算機の実例として以下ではMELCOM−1530

およびその改造形であるMELCOM−3100を用いる。この計算機は方式の発展段階としてはTRW社のAN−130という特殊用途の計算機を基礎に作られたもので，マイクロプログラミング方式としてはその後のIBM−360および370に至る過渡的なものとみることができる。しかしマイクロプログラミングの実現方式が前述のように機械語命令より下層での適用が一般的となつて発展したために，プログラムによりマイクロプログラムを随時交換することをソフトウエア構成の上で積極的に活用する動きがみられなかつた。MELCOM−1530はTRW−530を国産化したものであるが，ソフトウエアとしてはアセンブラを中心とした構成をとつており，FORTRAN，COBOLなどのコンパイラの実現はすべて三菱電機で実施されることになつた。

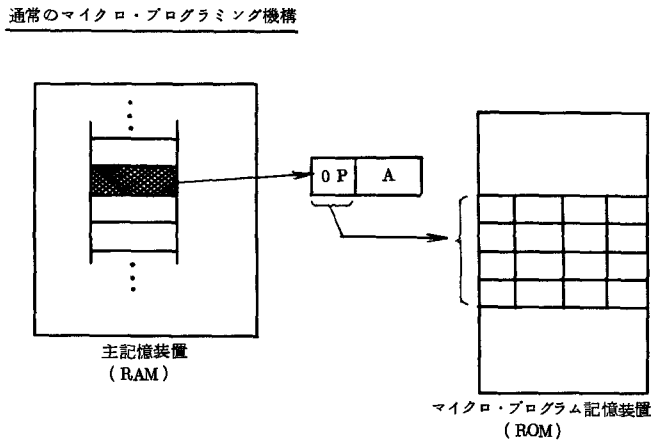


図 2.1

通常のマイクロプログラミング機構とストアドロジック形マイクロプログラミング機構との相違と簡単に図示すると図2.1のようになる。両者の特徴点をまとめると表2.1のようになる。ストアドロジック形ではマイクロ命令として特別なサイズ，記憶場所，処理対象を設けずに計算機ハードウエアの機構そのまゝを用いることが特色で，主記憶装置中にマイクロプログラム部分を常に抱え込んで処理が進められる。

表 2.1

	通常の マイクロ・プログラム機構	ストアド・ロジック形 マイクロ・プログラム機構
記憶場所	専用メモリ（ROM）	主メモリ
サイズ	長い（40～60bit）	語長
制御	専用ハードウエアによる	主プログラムと共用が中心
処理対象	ハードウエア・モジュール （ゲート，フリップフロップ等）	主メモリ，レジスタ内容
動作サイクル	ROM サイクル	主メモリ・サイクル
プログラム入換	困難	プログラムにより容易に可能

以下の議論の出発点としてMELCOM−1530および3100の基本的な構造について述べる。両者は中央処理装置の論理構造に関しては同等である。そのブロックダイアグラムは図2.2に示す通りで，基本要素として6個の18ビットのレジスタL，E，M，P，A，D，全加算器AD，半加算器AC，ケタ上げインディケータCおよびオーバフロウインディケータFを備えている。情報転送は原則として18ビット並列に行い，記憶装置の語構成も18ビットである。キャラクタ構成は6ビットであり，1語に3キャラクタを

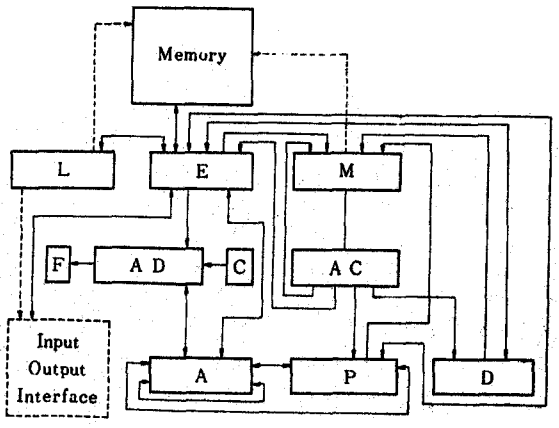


図 2.2

収納する。

機械語の命令は図 2.3 に示す構造をもつ。プライマリ・フィールドは基本命令コードとして使われる。アドレス・オプション・フィールドはレジスタの指定に使われる。カウント・オプション・フィールドはメモリのアクセスに際して番地変更の制御を行なうほか，シフト演算の語長の指定に使われる。メモリ・バンク・セレクションは関与するマイクロプログラムが貯えられる領域の指定の意味をもつ。セカンダリ・フィールドは，補助命令として使用するほか，特に演算がコアメモリの最初の 64 番地（この部分はスクラッチパッドと呼ばれレジスタに準ずる使い方をする）を対象とする場合にそのアドレスの指定に使用する。これらのフィールドのパターンの組合せによつてかなり多岐にわたる命令を持つことになる。

さて，このような機械語命令を用いてマイクロプログラムが作られ，各々のマイクロプログラムのもつ機能がアセンブラ語の機能として使われる。このためこの計算機のアセンブラ語の命令は通常の計算機のように機械語に直接対応するものでなく，それよりかなり水準の高いものとなる。アセンブラ語の命令とマイクロプログラムとのつながりを図示すると図 2.4 のようになり，アセンブラのオブジエクトプログラムはマイクロプログラムに対するコーリングシーケンスとなる。これはアドレス情報のみから成るのが通常の姿であり，リテラルが使われたときにそれが現われるが，いわゆる機械語命令はここには現われない。

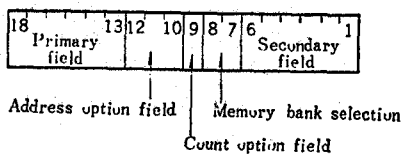


図 2.3

表 2.2　基本命令（一般）

Regular Primary Commands		
Octal Code	Sym. Code	Load Type Commands
60	NO	NO Operation
62	LA	Load A
63	LP	Load P
61	LD	Load D
54	LM	Load M
56	RA	Replace A
57	RP	Replace P
55	RD	Replace D
76	AP	Exchange A and P
71	ZA	Clear A
73	XA	Extract to A
72	MA	Merge to A
70	CA	Character to A
51	XE	Extract to E
52	ME	Merge to E
50	CE	Character to E
53	DX	Double Extract
67	*AS	Add Single Word
65	*AL	Add Least Significant Word
64	*AI	Add Intermediate Word
66	*AM	Add Most Significant Word
74	CS	Complement Set
75	CC	Complement Clear
77	CH	Complement Hold
Store Type Commands		
40	SE	Store E
41	SD	Store D
42	SA	Store A
43	SP	Store P
44	SM	Store M
45	HD	Hold D
46	HA	Hold A
47	HP	Hold P

表 2.3　基本命令（特殊）

Octal Code	Symb. Code	Primary Command
21	BR*	Branch
20	SK*	Skip
04	OL	Open Left Shift
06	OR	Open Right Shift
05	CL	Closed Left Shift
07	CR	Closed Right Shift
15	FL	Float Left
17	NR	Numeric Right Shift
16	MP	Multiply
14	DV	Divide
23	PA	Parameter Add
22	DC	Decimal Correct
31	MV	Move
32	CM	Character Move
33	PK	Pack
30	UP	Unpack
36	TB*	Table Search
37	MH*	Match
27	CO	Command Output
24	WI	Word Output
25	WO	Word Output
34	BI*	Block Input
35	BO*	Block Output
00	IT	Interrupt
02	TM	Terminate Interrupt

表 2.4　アドレス・オプション

Octal Code	Symbolic Code	Name
0	DM	Direct M
1	DL	Direct L
2	DP	Direct P
3	DD	Direct D
4	IM	Indirect M
5	IL	Indirect L
6	IP	Indirect P
7	ID	Indirect D

表 2.5　カウント・オプション

Binary code	Symbolic code	
0	C....Count	S....Single length
1	H....Hold	D....Double length

表 2.6　補　助　命　令

Secondary Commands		
Octal Code	Sym. Code	Name
60	NO	NO Operation
62	LA	Load A
63	LP	Load P
61	LD	Load D
76	AP	Exchange A and P
71	ZA	Clear A
73	XA	Extract to A
72	MA	Merge to A
70	CA	Character to A
67	AS	Add Single Word
65	AL	Add Least Significant Word
64	AI	Add Intermodiate Word
66	AM	Add Most Significant Word
74	CS	Complement, Set
75	CC	Complement, Clear
77	CH	Complement, Hold

表 2.7　命令のテスト条件

Octal Code	Symb. Code	Condition
00	NV	Never
01	UN	Unconditional
02	NS	Never Stop
03	US	Unconditional Stop
04	CY	Carry Indicator On
05	OV	Overflow Indicator On*
06	PY	Memory Parity Error Ind. On*
07	PW	Power Failure Ind. On*
10	SR	Status Request Ind. On*
11	MO	Momentary Alert Ind. On*
12	TY	Typewriter Alert Ind. On*
13	TR	Trace Indicator On
16	EB	End of Block**
17	EC	End of Card**
20	AP	A Positive
21	AN	A Negative
22	AD	A Odd
31	AZ	A Zero
23	EN	E Negative
32	NQ	E and A not Equal
33	EQ	E and A Equal
34	FL	A Field Low
35	FH	A Field High
36	NL	A Numeric Low
37	NH	A Numeric High

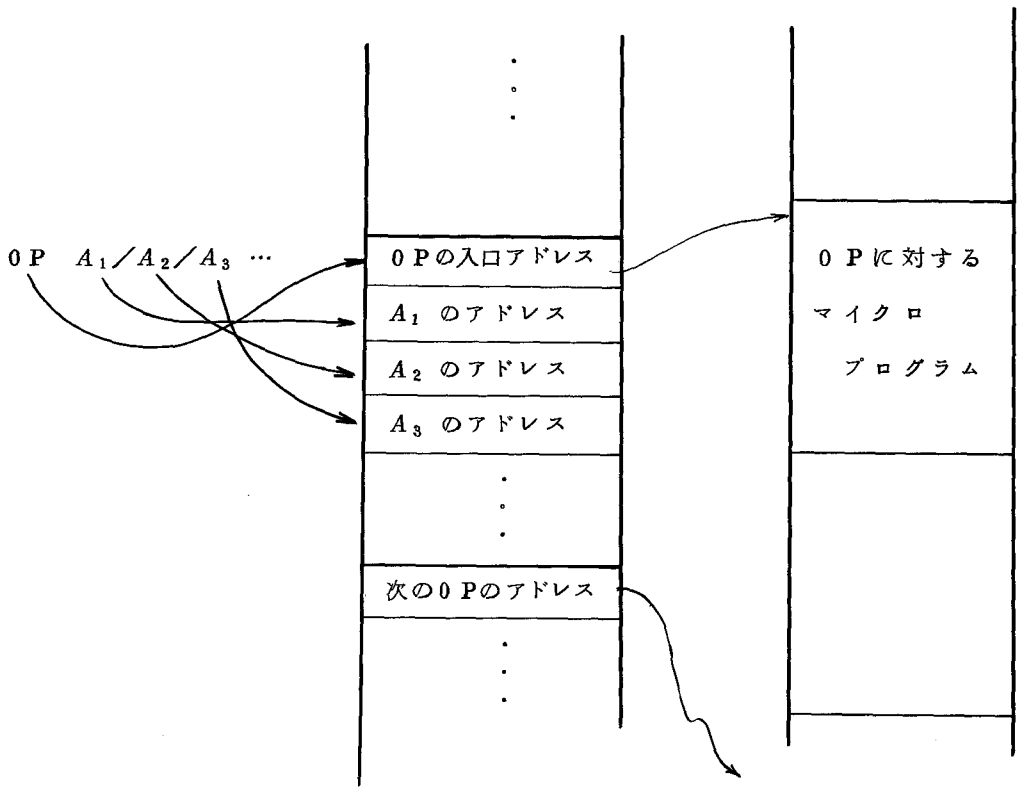

アセンブラ語の命令
アセンブラの
オブジェクトプログラム
(マイクロプログラムの
コーリングシーケンス)
マイクロプログラム
0P A_1／A_2／A_3 …
0Pの入口アドレス
A_1 のアドレス
A_2 のアドレス
A_3 のアドレス
次の0Pのアドレス
0Pに対する
マイクロ
プログラム

図 2.4

2.3 マイクロプログラミングによるオブジエクト計算機の構成

高レベルの手続き向き言語を用いる場合，ソース言語と計算機の構造とは一般にかなり隔つたものである。ソース言語はその目的からできるだけ個々の機種あるいはハードウエアに依存しない形で設定されるからこれは当然である。

ソース言語と計算機の構造との隔りを埋める方策として，次のものがある。

(1) ソフトウエアによる

a) コンパイラで全部機械語化する

b) 実行時体系を形成，充実する

(2) ハードウエアによる

(1)－a) はコンパイラの構成の中に機種固有の事情による処理が多く含まれるため，労力が大きくまたコンパイラの構成が機種ごとに異る要素が多く，標準化に逆行するため問題がある。現在では多かれ少なかれ実行時パッケージを形成して機械語そのまゝのオブジエクトプログラムとの併用を行なつているのが実態である。(1)－b) は機械の構造とは別に実行時の論理構造をソフトウエアで実現するものである。これには種々の形態があり，機械構造を殆ど使い若干の実行時ルーチンで補うものからまつたく新しい実行時処理体系をプログラムにより実現してインタープレータ方式に近い形をとるものまである。ソフトウエアを用いると融通性はあるが速度効率の低下をまぬがれない。(2)のハードウエアによる方法はソース言語を直接実行する計算機として以前から検討されているが，速度効率は向上するけれどもコスト面で不利になることと種々な言語，データ形式に対する融通性が減ることが問題である。

(1)－b) の実現にマイクロプログラミングを適用すれば，融通性を保ちつつ速度効率の低下をある程度低減できると期待される。ストアドロジック形マイクロプログラミング方式は，ソース言語ごとに適当なマイクロプログラムを作成し，交換して使用できるので，この用法でその効用を発揮するものである。

この場合，実行時体系は

a) 標準マイクロプログラム

b) オブジエクト用マイクロプログラム

c) 実行時ライブラリ

から成る形をとることになる。b) は特定の言語に固有なマイクロプログラムで，コンパイラのローディングの段階で機械に入る。c) はやはり言語に固有であるが，例えば入出力ルーチンのような共通的なものと関数サブルーチンのような問題に依存するものとがある。

オブジエクト計算機の構成にあたつては，その機能単位の大きさの選定が現実に問題となる。これについては統一的な基準は未だないといつてよく，ソース言語の形と性質，計算機の処理機能単位，記憶容量などの関連で経験的にきめているのが現状である。

2.4 COBOL コンパイラにおける構成例

ここでMELCOM−1530および3100のCOBOL コンパイラを例にとつて，マイクロプログラミングによるオブジエクト計算機の構成を検討する。

2.4.1 COBOL の文法

COBOL の文法の特徴を要約すると次のようになる。

(1) １０進可変桁のデータ，可変長のリテラルを基礎としている。

(2) 算術演算は簡単なもののみであるが，可変桁データの移動，つき合わせ，編集などの事務用処理が豊富にある。

(3) 英語のキイワードを用いた英語様の文を基礎とし，文章の原形から選択的に必要項を記述する形（Alternatives and Options）の文法をもつ。

(4) データが階層構造をもつ。

(5) プログラムの構造も階層的である。

これらのうち，マイクロプログラミングによるオブジエクト計算機の構成の上で問題となるのは主に(1)，(2)である。

(3)はコンパイルの方法に関連する。COBOL の文法の一部を示すと，例えば加算の文は次の形の規則をもつ。

$$\mathrm{ADD}\left\{\begin{array}{l}\mathrm{literal\text{-}1}\\ \mathrm{data\text{-}name\text{-}1}\end{array}\right\}\left[\left[\left\{\begin{array}{l}\mathrm{literal\text{-}2}\\ \mathrm{data\text{-}name\text{-}2}\end{array}\right\}\cdots\right]\right.$$

$$\left.\left[\left\{\begin{array}{l}\mathrm{TO}\\ \left\{\begin{array}{l}\mathrm{literal\text{-}n-1}\\ \mathrm{data\text{-}name\text{-}n-1}\end{array}\right\}\mathrm{GIVING}\end{array}\right\}\right]\right]\ \mathrm{data\text{-}name\text{-}n}\ .$$

ここで，{ }はどちらか一つをとることを示し，〔 〕は必要に応じ書けることを示す。literal は定数，data-name は変数名を示し，ADD,TO,およびGIVING はキイワードである。プログラムに現われる加算文としては，

ADD　A　B

ADD　A　TO　B

はAとBとを加えて結果をBに残すこと，

ADD　12.5　C　GIVING　D

は12.5 をCに加えて結果をDに残すことをそれぞれ示す。A,B,C 等の変数名は別の部分（DATA DIVISION）で宣言されており，その構造についても桁数，小数点の位置がそれぞれ明確に指定されている。更に，変数名が他の変数名の修飾をうけることもあり，配列の要素であることも許される。

別の例として，データ移動の文は次の規則をもつ。

$$\mathrm{MOVE}\left\{\begin{array}{l}\mathrm{data\text{-}name\text{-}1}\\ \mathrm{CORRESESPONDING\ data\text{-}name\text{-}2}\\ \mathrm{literal}\end{array}\right\}$$

$$\mathrm{TO\ data\text{-}name\text{-}3}\quad\left[\mathrm{data\text{-}name\text{-}4}\right].$$

簡単な形としては

MOVE　A　TO　B

によりデータAをBに移す操作を行なうのであるが，このとき，A，Bがどのように宣言されているかによつて種々の処理が挿入されたり，

桁合わせのしかたが異なつたりする。データが数値か文字列か，あるいは集団名か，または出力形式かなどの指定が宣言の部分で指定されている。またdata-name-1としてZERO，SPACESと書くことによつてdata-name-3をクリアすることも許される。CORRESPONDINGは集団名に対して適用されるもので，各々に属するデータのうち名前が一致するものだけを移動処理の対象とする機能をもつ。例えば

Aが　S，T，U，V，W，X，から成り

Bが　X，Y，Z，S，R，Q，から成る

とすると，

MOVE CORRESPONDING A TO B.

によつて，Aの中のSとXがBの中のSとXに移される。

このように，FORTRAN，ALGOL と異つて，COBOL ではデータの種類と構造が多岐にわたることが特徴で，これがオブジエクト計算機の構成を強く方向づけることになる。COBOL 向きに設計されたハードウエアを用いる場合でなければオブジエクト計算機の構成は複雑になることは避けられない。FORTRANのようにデータ形式がREALとINTEGERに限定され，それらがハードウエアの浮動小数点数と固定小数点数に対応している場合に比べるとこの問題は容易に推察されるであろう。

2.4.2　COBOL オブジエクト計算機

前述のようなCOBOL の文法によるプログラムを2進形計算機MELCOM-1530／3100 で処理するために，そのマイクロプログラミング機構を用いてCOBOL 用のオブジエクト計算機を構成した。本節以降でその構成について記述する。

まず，可変桁10進演算に対処するために，演算機構として

アキユムレータ	10進30桁，符号
バツフア・アキユムレータ	10進30桁，符号
溢れインディケータ	1語長

TALLY エリア　　１０進５桁

論理アキユムレータ　　１語長

を設けた。これらは主記憶装置の先頭領域にあるスクラッチパッドの中にとる。

アキユムレータおよびバッファ・アキユムレータは１０進データの四則演算に使用するほか，桁合わせを伴う数の移動の過程でも使用する。COBOL で使用する数値は最大桁数１５桁であるから，この２個のアキユムレータは中央に小数点を置き整数部小数部ともに１５桁とる使い方を基本とする。メモリとアキユムレータの間の移動の際に位取りが適当に行われるよう，アキユムレータ内のどの位置に数値の先頭が入るかを指定する形でLoad，Storeの命令を構成する。この位取りのための桁位置情報は，ソースプログラム中でデータ記述に与えられている桁数の情報をもとにしてコンパイラで処理をしてオブジエクトプログラムに出す方法をとつた。２個の数の演算は，両者を２個のアキユムレータに入れた後アキユムレータ間の演算として処理する。

溢れインデイケータは演算過程で溢れが発生したことを示すためのもので，演算をプログラムによつて構成したためにこのインデイケータのセット，リセットもプログラムで行われる。

COBOL にはデータ移動の過程でデータ中に含まれる特定文字の個数をカウントしたり，頭のゼロ（leading zero）を空白に変えながらその桁数を調べたりする操作が何種数かあり，かなり基本的な処理として使用される。これに対処して作られたのがTALLY エリアであり，TALLY として求めた値をプログラムで使用するためにこれは結果を１０進形で残している。

論理アキユムレータは論理判断による分岐の制御などに使用するためのもので，これは直接プログラムの表面に出ることはないから，この計算機で扱いやすい１語長という大きさをもたせてある。

次にこの演算機構を用いてCOBOL プログラムの実行を行なうため，マイクロプログラムルーチンを一揃い設けた。最終版であるMELCOM−

3100 Mk−Ⅱ／Ⅲ COBOLではこのルーチンの総数は77であり，その内訳は次のようになつている。

算術演算	13
データ移動	17
条件判定	25
手続き分岐	5
論理処理	7
その他	10

これらのマイクロプログラムルーチンの機能については本章付録にまとめた。

マイクロプログラムルーチンの数は多いが，よく似たもの，共通的な処理を含むものがかなりあり，機能ブロック毎にまとめて入口点だけを複数個設ける方法をとつたので，全体として比較的コンパクトにまとめ得た。総ワード数約600である。

このような構成のオブジエクト計算機を使用することによつて，規模の大きくない計算機を用いてCOBOLの文法仕様をかなり大きく，殆どFull仕様に近いものとすることが可能となつた。文献(7)に示された各種計算機のCOBOLコンパイラの比較でわかるように，9種の計算機のほゞ中位に当る規模をもつMELCOM−1530および3100で，最大級の仕様を実現しており，例題のコンパイル所要時間も中位以上に入つている。

2.4.3 算術演算マイクロプログラムの用法

COBOLの算術文に対応するオブジエクトプログラムの構成とその中で使われるマイクロプログラムルーチンについて，加算を例にとつて説明する。

ソース言語での加算の規則は次の形をもつている。

```
ADD { literal-1   } [ { literal-2   } ... ]
    { data-name-1 }   { data-name-2 }

[ { TO                          } ]
[ { literal-n-1                 } ]  data-name-n.
[ { {                 }GIVING   } ]
[ { data-name-n-1               } ]
```

これに対応して，オブジェクトプログラムの基本形を次の様に構成する。

```
LOA     C1／C2／n1
LOB     C3／C4／n2
ADC
LOB     C5／C6／n3      ┐
ADC                     ┘ 加算 data-name
 .                        が複数個のとき
 .
 .
RUD                     ←オプション
SNA     Ci／Ci+1／nj    ←STR を使うこともある
SZE     W1              ←オプション
 .                      ┐
 .                      ┘ ON SIZE ERROR
 .                        があるとき
```

W_1　NOP

ここでは読みやすさのためにこの計算機のアセンブラ語の形で表現したが，実際にはこれに対応するリロケータブル・オブジェクトプログラムの形で表現されている。LOA，LOB，ACD等はマイクロプログラム名であるが，これらはその入口の位置の情報で示されている。上から3行はソース文の最初の2個のliteral またはdata-name に対応し，ソース文が多くの項の加算を指定しているときに4行目以降にLOBとADC の対が必要なだけ挿入される。RUD は加算結果に丸めを施すマイクロプログラムで，COBOL ではこの機能はオプションとして使用するから，必要なときにこのRUD を挿入する。SNA は加

算結果のストアであるが，これはストア先が数値項目と指定されている場合に使用する。結果を出力項目（Reportic Item）としてストアするときにはSNAの代りにSTRを用いる。溢れインデイケータがセットされているか結果の桁の上落ちがある場合にはこのルーチンでは実際のデータ転送は行わない機構になつている。溢れの検出に伴う処置はソース文にON SIZE ERROR オプションを書いて指定するが，それがあるときはSZE 以降のものが挿入される。NOP はSZE があるときにソースプログラムの次のステツプへ飛ぶために使う他はノーオペレーションである。主要なルーチンの機能を簡単にまとめると次の通りである（詳細は付録）。

LOA： C_1 から n_1 桁をアキユムレータの C_2 からへClear Add，同時に溢れインデイケータをクリア。

LOB： C_3 から n_2 桁をバツフア・アキユムレータの C_4 からへClear Add.

ADC： アキユムレータとバツフア・アキユムレータを加算し，結果をアキユムレータに残す。溢れが発生すれば溢れインデイケータをセット。

SNA： アキユムレータの C_i からの n_j 桁を C_{i+1} からの位置へ転送。

アキユムレータおよびバツフア・アキユムレータは中央に小数点を置き上下各１５桁として作用する。そのためADCによる加算に際してはデータの桁移動は起らないが，前後のロード，ストアで桁位置調整を行なう。COBOLではデータの構造は宣言部で詳しく記述されているから，その情報を用いてコンパイラで桁移動の指定をすることができる。それに基いて上述の C_1，C_2 などが与えられる。この計算機は語の中の桁位置を指定した処理が効率よく出来る機構となつているので，このような桁単位の処理を含むオブジエクト計算機を無理なく実現できた。

前述の各種オプションに伴うルーチンの挿入指定およびデータの個

数に依る反復挿入はコンパイラによつて行われる。

減算においては，ADC が減算ルーチンSBC に替るほかは同一である。また乗除算もほゞ似た形をとる。

2.4.4 データ移動マイクロプログラムの用法

データ移動の文は単純な形をもつが，処理がデータ自体の性質により多くの類に分かれるので，やゝ複雑になる。

データ移動のソース文の規則は次の通りである。

$$\text{MOVE}\left\{\begin{array}{l}\text{data-name-1}\\ \text{CORRESPONDING data-name-2}\\ \text{literal}\end{array}\right\}$$
$$\text{TO data-name-3 〔data-name-4〕.}$$

CORRESPONDINGの処理はコンパイラで行なうから，オブジエクトプログラムに現われるのは1個のデータ移動のみである。

データの種類の大別として，数値データ(N)と文字データ(AN)がある。AからBへのデータ移動に際してAとBの桁数が異るとき，NとANで扱いに差がある。数値データは小数点位置を合わせることが基本となるが，文字データは左端を合わせることが原則である。数値データと文字データの相互の転送の場合は，転送先の形式に従う方法がとられる。転送先の桁数が大きい場合，Nならば余分の位置に0を埋めるべきでありANならば空白(△)を埋めるべきである。また文字データの場合でも特に右端を合わせたいことがあり，そのとき JUSTIFIED RIGHTと指定される。また，出力項目(Reportic Item)へデータを移すときには編集処理が伴うからやゝ異つた処理となる。ZERO, SPACE といつた記法，およびALL literal 記法の場合，転送先の各桁に同一文字を反復送り込む。最後に集団項目(Group Item)の場合，ANとみなすことにする。

以上の変化の組合せに対処するため，データ移動のマイクロプログラムルーチンを7個設けた。その詳細は付録にあるので，ここでは処理内容を一覧するための図を掲げるに止める(図2.5)。

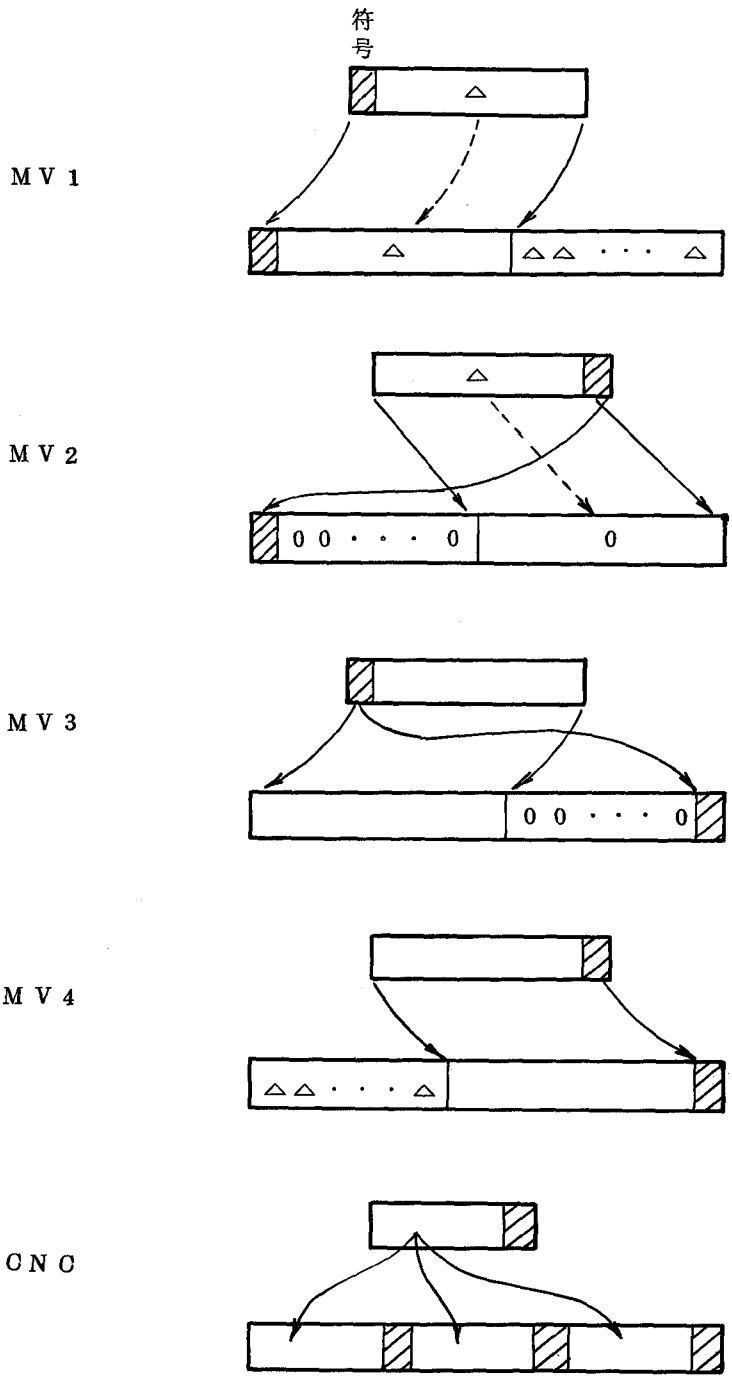


図 2.5

転送元と転送先の性質の組合せによりどのルーチンを使用するかを表2.8にまとめた。この表の基本規則は次のように決めてある。

(1) ＡＮ→ＡＮ　　ＭＶ１

左からつめ，余白には空白を入れる。

(2) ＡＮ→Ｎ　　ＭＶ２

右に揃える。空白と上位余白には０を入れる。符号位置を左端へ移す。

(3) Ｎ→ＡＮ　　ＭＶ３

左に揃える。余白には０を入れる。符号位置を右端へ移す。

(4) Ｎ→Ｎ　　ＳＮＭ

小数点を合わせる。両端の余白には０を入れる。符号は左端に入れる。

(5) Ｎ→Ｒｅｐｏｒｔ　　ＭＶＲ

小数点を合わせ，編集を行なう，余白には０を入れる。

(6) ＡＬＬ（ＳＰＡＣＥ，ＱＵＡＴＥ）→ＡＮ　ＣＮＣ

転送先の桁数だけ繰返して移す。

(7) ＡＮ→ＡＮ　　ＭＶ４

右に揃える。余白には空白を入れる。

なお，すべての場合について転送先の桁数の関係ではみ出した部分は切捨てる。表2.8の中で示した番号は上の規則の番号である。

この処置によりデータ移動を７種の基本機能で実現することができたが，これらは共通の処理が多いので実際にはルーチンの個数は少く，例えばＭＶ１～ＭＶ４の４種は１個のルーチンに４個の入口を設ける形で実現して，マイクロプログラムルーチンのメモリ占有量を減らしている。

表2.8

移動先 / 情報源	Group Item	A／N	N	Reportic Item	Justified Right
Group Item	(1) MV1	(1) MV1	——	——	(7) MV4
A／N	(1) MV1	(1) MV1	(2) MV2	——	(7) MV4
N	(1) MV1	(3) MV3	(4) SNM	(5) MVR	(3) MV3
Reportic Item	(1) MV1	(1) MV1	——	——	——
ZERO，ZEROS	(1) MV1	(3) MV3	(4) SNM	(5) MVR	(3) MV3
SPACE，SPACES	(6) CNC	(6) CNC	(6) CNC	(6) CNC	(6) CNC
QUATE，QUATES	(6) CNC	(6) CNC	(6) CNC	(6) CNC	(6) CNC
ALL非数リテラル	(6) CNC	(6) CNC	(6) CNC	(6) CNC	(6) CNC

2.5　事務用問題向き言語への適用例[8]

前節で述べたCOBOLオブジェクト計算機は2進処理を基礎とするハードウエアの上に可変桁10進の処理系を実現し，加えて事務計算に頻出する処理の要素となる機能を備えている。これらの機能が一群のマイクロプログラムルーチンとしてまとめられているわけであるから，COBOLと同系統の処理を行なうときには活用できる。

その一つの例として，コンパイラ言語より更に適用対象に密着した形で構成された，いわゆる問題向き言語に属するRPG形の言語のプログラムの実行段階にやはりオブジェクト計算機として使用することが考えられた。MELCOM−3100の場合はRPGを特にACE（Automatic Compiling Editor）と呼んでいる。この形の言語は事務計算が殆どの場合入力データの集計その他の処理の結果を報告書の形で出力するという流れで構成されていることに着目してプログラムを簡易化する工夫をしており，計算処理の指定，入力データの形式指定，出力データの形式指定を独立した様式で穴埋め式（Fill−in−the−Blank）に近い定形記法で書かせようとしている。データの記述を複雑にできるよう考えるとファイル記述などが入るため様式の種類が増える。代表的なIBMとRPGの場合は全部で6種の様式があるが，単純な処理パターンであればそのうち3種か4種で充分プログラムが作れる。MELCOM−3100 ACEでは様式を4種類設けた。

MELCOM-ACE　CODING FORM A　　プログラム名＿＿＿＿＿＿＿＿

ファイル明細書

001　ページ（全 4 ）　　作成者名＿＿＿＿＿＿＿＿　作成日付　年　月　日

ページ	行番号	カード区分(F)	ファイル名	入出力装置 記号	入出力装置 番号	印刷最大行数(注1)	入出力区分(I・O)	処理終了桁(E)	ブロックのレコード数(注2)	レコードの字数	標準ラベル指定(S)	標準ラベルの指定 ファイル名	標準ラベルの指定 ファイル番号	標準ラベルの指定 保存日数	EOF標識(注3)	データ制御(K)	レコード形式(U)	変換テーブル番号(注5)	(注6)	業務記号
1 2 3 4 5 6	7	8 9 10 11 12	13 14 15 16		17 18 19 20	21	22	23 24 25	26 27 28 29 30	31	32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49			50	51	52	53 54 55 56 57	58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80	
001	010	F	CARD	CR					60					E1					TEST−I	
	020	F	TAPE1	TU	33		I		100		81	SKASAN−DATA	001							
	030	F	PRINTLP																	
	040	F																		
	010	F																		

	8	磁気テープ	ラインプリンタ	カードリーダ	カードパンチ	紙テープリーダ	紙テープパンチ
入出力装置		TU	LP	CR	CP	PR	PP
印刷最大行数(注1)			○				
入出力区分(I・O)						○	○
処理終了桁(E)						○	○
ブロックのレコード数(注2)							
レコードの字数		○	○	○	○	○	○
標準ラベル指定(S)		○					
EOF標識(注3)		○					
データ制御(K)						○	○
レコード形式(U)						○	○
変換テーブル番号(注5)							

○印は必須，空白部は任意，その他の部分は書いてはならない。
(注1) 途中の空白行も含む1ページ分(書かなければ60とみなす)。
(注2) 書かなければ1とみなす。
(注3) E1〜E9以外を書いてはならない。
(注4) 8(紙テープ8単位コード)，6(紙テープ6単位コード)，E(磁気テープ偶数パリティ)のうちいずれかを書くことができる。
(注5) システム・テープに登録されたPTCTライブラリィの登録番号(3桁の数字)を書く。
(注6) カラム58〜72は空白にしておくこと。
←は左つめ，→は右つめを表わす。コメント行にはカラム8に＊を入れる。

図2.6(a)　ファイル明細書

MELCOM-ACE CODING FORM B

プログラム名＿＿＿＿＿＿＿＿＿＿

002 ページ（全 4 ）

入力レコード明細書

作成者名＿＿＿＿＿＿＿＿＿＿ 作成日付 年 月 日

カード番号 ページ	行番号	カード区分(注1)	ファイル名	読取条件(注1)	レコード選択(注2)	項目指定 名前	位置 始点	位置 終点	小数点位置	加算指定 宛先名	桁数(注3)	コントロールレベル(注4)	比較項目(注5)	比較条件(注6)	レコード選定 基準値	(注7)	業務記号
002	010	I	P1CARD														TEST-1
	020	I				TEMP1	1	60		KASA1	9						
	030	I				TEMP2	8	130		KASA2							
	040	I				TEMP3	15	240		KASA3							
	050	I				TEMP4	26	290		KASA4	6						
	060	I				TEMP5	31	370		KASA5							
	070	I	P2TAPE1														
	080	I				TEMPA	1	60		KASAA	9						
	090	I				TEMPB	8	130		KASAB							
						TEMPC	15										
	150																

	カード番号	ファイル名	レコード選択	項目指定	加算指定	レコード選定
ファイル名の行	○					
レコード選定の行			○	○	○	○○
項目指定の行				○	○	

○印は必須，空白部は任意，その他の部分は書いてはならない。

(注1) P1～P9以外を書いてはならない。

(注2) 01～99(10位の0は必要)以外を書いてはならない。

(注3) 書かなければ項目指定の桁数と同じとみなす。

(注4) L1～L9以外を書いてはならない，必らずL1から順に使用すること。

(注5) K1～K9以外を書いてはならない。

(注6) E(等価)N(基準値外)G(基準値より大)L(基準値より小)。

(注7) カラム60～72は空白にしておくこと。

←は左づめ，→は右づめを表わす。

コメント行にはカラム8に＊を入れる。

図 2.6(b) 入力レコード説明書

MELCOM-ACE CODING FORM C

プログラム名＿＿＿＿＿＿＿＿＿＿

003 ページ（全 4 ）

処理手続書

作成者名＿＿＿＿＿＿＿＿＿＿ 作成日付 年 月 日

カード番号 ページ	行番号	カード区分(P)	処理レベル(注1)	処理条件(注2) および	項目1 リテラル 項目名	操作コード	項目2 リテラル 項目名	宛先指示 項目名	宛先指示 字数	宛先指示 小数点位置	宛先指示 符号(注3)	結果標識(注2) 計算結果 正 1>2	結果標識(注2) 計算結果 零 1=2	結果標識(注2) 計算結果 負 1<2	結果標識(注2) 比較結果 1>2	結果標識(注2) 比較結果 1=2	結果標識(注2) 比較結果 1<2	桁送れ標識(注2)	(注4)	業務記号
003	010	P	LS			DISPLY@TEST-1 START@														TEST-1
	020	P	LS			OFF														
	030	P	LS			ON														
	040	P	LIE1			ON														
	050	P	LIE1		KASA1	SUB	TEMP1													
	150	P																		

(注1) 冒頭処理はLS，細目処理は空白，合計処理はL1～L9，LR，しめくくり処理はLLである。

(注2) 01～99，L1～L9，LR，P1～P9，KG，KE，KL，OF，以外を書いてはならない。

(注3) 加減乗除算の符のみ書くことができる。

(注4) カラム70～72は空白にしておくこと。

←は左づめ，→は右づめを表わす。 コメント行にはカラム8に＊を入れる。

図 2.6(c) 処理手続書

MELCOM-ACE CODING FORM D

プログラム名＿＿＿＿＿＿＿＿＿＿

004 ページ 全 4 ）

出力レコード明細書

作成者名＿＿＿＿＿＿＿＿＿＿ 作成日付 年 月 日

カード番号 ページ	行番号	カード区分(O)	ファイル名	出力の時期(H,T)	出力条件(注1) および	ラインスキップ 印刷前	ラインスキップ 印刷後	改行 印刷後	項目名	始点	出力抑止(E)	編集様式(注2)	リテラルまたは編集様式	(注3)	業務記号
004	010	O	PRINT												TEST-1
	020	O		T	LR	0	5								
	030	O								31			@KASAN-CHECK LIST CARD@		
	040	O		T	LR		3								
	050	O							KASA1	10					
	060	O							KASA2	20					
	070	O							KASA3	30					
									KASA4	50					
	150														

	カード番号	ファイル名	出力条件	項目名	始点	出力抑止	編集様式	リテラルまたは編集様式
ファイル名の行	○							
レコード 指定の行	LP その他	○						
項目指定の行				○	○			

○印は必須，空白部は任意，その他の部分は書いてはならない。

(注1) 01～99，L1～L9，LR，P1～P9，E1～E9，KG，KE，KL，OF以外を書いてはならない。

(注2) ZZ，ZZ9.9一の形(15桁以内であること。桁数は定義時に従う。

(注3) カラム66～72はブランクにしておくこと。

その他 非数値項目(小数点位置が0～9以外)を編集することはできない。

←は左づめ，→は右づめを表わす。

コメント行にはカラム8に＊を入れる。

図 2.6(d) 出力レコード明細書

この形の言語は，COBOL のような手続き向きの言語と比べると文の形の変種が無いに等しく定形記法でプログラムされるため，解析が極めて簡単となる。処理内容が可変桁１０進事務系であるから，それに適したハードウエア設計を伴えば問題はあまり無くなるが，２進の語構成のハードウエアでは実行時の処理に手間どることは避けられない。そこでCOBOL の場合に効果的であつたオブジエクト計算機をほゞそのまゝ採用して，このマイクロプログラムルーチンを呼ぶ命令列を ACE のプロセツサから生成することにした。ACE の文法機能はCOBOL よりも簡単であり，COBOL にみられる特殊なオプシヨンは不要であるものが多いが，基本的な１０進可変桁の処理，報告書作成のための出力書式制御および編集処理などの機能においては殆ど同形式であるため，COBOL のために作られたオブジエクト計算機がよく適合性を示した。

２６　結　　言

マイクロプログラムを主記憶装置に置き，プログラム制御で随時交換可能なストアドロジツク形の計算機において，ソース言語に適合したマイクロプログラムルーチンを用意して実行時に動作させることによつてオブジエクト計算機を実際のハードウエアの構能よりも大幅にソース言語の性格に近づけ得ることに着目し，２進の語構成を基礎構造とする計算機で１０進可変桁の事務処理を目的とするCOBOL に適合するオブジエクト計算機を構成した例について述べた。

第2章　付　　録

COBOL マイクロプログラムルーチンの機能

凡　例

U－Acc	アキュムレータの上半分
L－Acc	アキュムレータの下半分
U－Buf	バッファアキュムレータの上半分
L－Buf	バッファアアキュムレータの下半分

上下を一括して扱うときは単にAcc,Buf と記す。

(OV)	溢れインディケータ
(LA)	論理アキュムレータ
(TALLY)	TALLY エリア

命　令　形　式	機　　能
LOA　C_1／C_2／n_3	Load Accumulator U−Acc，L−Acc および（ＯＶ）をクリアし，アドレス C_1 で始まる n_3 桁の数項目の内容をAcc中のアドレスからの位置に移す。
LOB　C_1／C_2／n_3	Load Buffer Accumulator U−Buf，L−Buf をクリアし，アドレス C_1 で始まる n_3 桁の数項目の内容をBuf上のアドレス C_2 からの位置に移す。
ADC	Add U−Acc，L−Acc の内容と U−Buf，L−Buf の内容を加えて結果をAccに残す。 溢れが発生すれば（OV）をセットする。
SBC	Subtract U−Acc，L−Accの内容から U−Buf，L−Buf の内容を引いて結果をAccに残す。 溢れが発生すれば（OV）をセットする。
MLT　C_1／C_2／n_3	Multiply L−AccとL−Buf の内容の積を作り，結果をBufに残す。その後Buf上のアドレス C_1 から n_3 桁をAcc上のアドレス C_2 からの位置に移す。

命令形式	機能
DVD	Divide U－Acc，L－Accの内容をL－Buf の内容で割つて結果をAccの下位１６桁に残す。Accの上位１４桁には余りが入る。 次の場合に(OV)をセットする。 (1) 商が１７桁以上になる。 (2) Accの内容に数値以外の桁がある。
RUD C_1	Round Acc上のアドレスC_1に数値５を加える。 この結果溢れが発生すれば(OV)をセットする。
SNA $C_1/C_2/n_3$	Store Numeric Arithmetic U－Acc，L－Acc上のアドレスC_2からn_3桁をアドレスC_1からの位置へ移す。 次の場合は移動を行わない。 (1) (OV)がセットされている。 (2) Acc内のC_2より上方にゼロでない桁がある。 (このとき(OV)をセットする。)
STR $C_1/C_2/C_3$	Store Reportic U－Acc，L－Acc上のアドレスC_2からの内容を，C_3から始まる位置にある編集情報に対応させながら，C_1から始まる位置へ移す。
SZE W_1	Size Error (OV)のテストを行ない，ゼロならばW_1に飛ぶ，ゼロでなければ次の文の実行に移る。

命　令　形　式	機　　能
ACS　n_1	Accumulator Shift U−Acc，L−Acc の内容を n_1 桁シフトする。 $n_1>0$ならば左に，$n_1<0$ならば右シフトする。
BUS　n_1	Buffer Accumulator Shift U−Buf，L−Buf の内容を n_1 桁シフトする。 $n_1>0$ならば左に，$n_1<0$ならば右にシフトする。
ACL	Buffer Accumulator Clear U−Buf，L−Buf および（OV）をクリアする。
MV1　$C_1/C_2/n_3/n_4$	Move C_1 から始まる n_3 桁を C_2 から始まる n_4 桁の項目へ左につめて移す。$n_3>n_4$ ならば n_4 桁より右を切り捨て，$n_3<n_4$ ならば余白には空白桁を移す。
MV2　$C_1/C_2/n_3/n_4$	Move C_1 から始まる n_3 桁を C_2 から始まる n_4 桁の項目へ右につめて移す。$n_3>n_4$ ならばはみ出した分を切り捨て，$n_3<n_4$ ならば余白にはゼロをつめる。
MV3　$C_1/C_2/n_3/n_4$	Move C_1 から始まる n_3 桁を C_2 から始まる n_4 桁の項目へ左につめて移す。$n_3>n_4$ ならばはみ出した分を切り捨て，$n_3<n_4$ ならば余白にゼロをつめる。
MV4　$C_1/C_2/n_3/n_4$	Move C_1 から始まる n_3 桁を C_2 から始まる n_4 桁の項目へ右につめて移す。$n_3>n_4$ ならばはみ出し分を切り捨て，$n_3<n_4$ ならば余白に空白桁を入れる。

命　令　形　式	機　　能
CNC　C_1／C_2／n_3／n_4	Copy n Characters C_1 から始まる n_3 桁を C_2 から始まる n_4 桁の項目へ C_1 側の内容を繰返して発生させつゝ移す。
SNM　C_1／C_2／n_3	Store Numeric Move U－Acc，L－Acc 上のアドレス C_2 から n_3 桁を C_1 から始まる位置へ移す。（OV）には無関係
MVR　C_1／C_2／C_3	Move Reportic U－Acc，L－Acc 上のアドレス C_2 からの内容を，C_3 から始まる位置にある編集情報に対応させながら，C_1 から始まる位置へ移す。（OV）には無関係
TAR　C_1／n_2／C_3／C_4	Tally Au Replacing C_1 から始まる n_2 桁のうちに C_3 で示される文字と同じものがあれば C_4 で示される文字に置きかえ，その個数を（TALLY）に入れる。
TUR　C_1／n_2／C_3／C_4	Tally Until First Replacing C_1 から始まる n_2 桁のうちに C_3 で示される文字と同じものが現われるまで C_4 で示される文字に置きかえ，その個数を（TALLY）に入れる。
TLR　C_1／n_2／C_3／C_4	Tally Leading Replacing C_1 から始まる n_2 桁のうちに C_3 で示される文字以外の文字が現われるまで C_4 で示される文字に置きかえ，その個数を（TALLY）に入れる。
TAC　C_1／n_2／C_3	Tally All C_1 から始まる n_2 桁の中に C_3 で示される文字と同じものがあればその個数を（TALLY）に入れる。

命　令　形　式	機　　能
TUC　C_1／n_2／C_3	Tally Until First C_1 から始まる n_2 桁の中に C_3 で示される文字と同じものが現われるまでの桁数を（TALLY）に入れる。
TLC　C_1／n_2／C_3	Tally Leading C_1 から始まる n_2 桁の中に C_3 で示される文字以外のものが現われるまでの桁数を（TALLY）に入れる。
RAC　C_1／n_2／C_3／C_4	Replace All C_1 から始まる n_2 桁の中にある C_3 で示される文字をすべて C_4 で示される文字に置きかえる。
RUC　C_1／n_2／C_3／C_4	Replace Until First C_1 から始まる n_2 桁の中に C_3 で示される文字と同じものが現われるまで C_4 で示される文字に置きかえる。
RLC　C_1／n_2／C_3／C_4	Replace Leading C_1 から始まる n_2 桁の中に C_3 で示される文字以外のものが現われるまで C_4 で示される文字に置きかえる。
RFC　C_1／n_2／C_3／C_4	Replace First C_1 から始まる n_2 桁の中に初めて現われる C_3 で示される文字と同じものを C_4 に置きかえる。

命　令　形　式	機　　能
	U－Acc，L－Acc とU－Buf，L－Buf とを比較して下記の条件を満足すれば（LA）を１に，満足していなければ（ＬＡ)を０にする。
NGC	Numeric Greater than　(Acc) ＞ (Buf)
NGN	Numeric Not Greater than　(Acc) ≦ (Buf)
NEC	Numeric Equal to　(Acc) ＝ (Buf)
NEN	Numeric Not Equal to　(Acc) ≠ (Buf)
NLC	Numeric Less than　(Acc) ＜ (Buf)
NLN	Numeric Not Less than　(Acc) ≧ (Buf)
	C_1 から始まる n_2 桁を C_2 から始まる n_3 桁と比較して下記の条件を満足すれば（LA）を１に，満足していなければ（LA）を０にする。
AGC　$C_1/C_2/n_3/n_4$	Non-Numeric Grecter than　$(C_1) > (C_2)$
AGN　$C_1/C_2/n_3/n_4$	Non-Numeric Not Grecter than　$(C_1) \leq (C_2)$
AEC　$C_1/C_2/n_3/n_4$	Non-Numeric Equal to　$(C_1) = (C_2)$
AEN　$C_1/C_2/n_3/n_4$	Non-Numeric Not Equal to　$(C_1) \neq (C_2)$
ALC　$C_1/C_2/n_3/n_4$	Non-Numeric Less than　$(C_1) < (C_2)$
ALN　$C_1/C_2/n_3/n_4$	Non-Numeric Not Less than　$(C_1) \geq (C_2)$
CND　$C_1/C_2/n_3$	Test Condition Variable C_1 から始まる n_3 桁と C_2 から始まる n_3 桁を比較して等しければ（LA）を１に，等しくなければ（LA）を０にする。

命　令　形　式	機　　能
	C_1 から始まる n_2 桁がAlphabetic かNumeric かを調べ，次の条件を満足すれば（LA）を1に，満足しなければ（LA）を0にする。
CLA　C_1／n_2	IF Class is Alphabetic　(C_1)：Alphabetic
CLN　C_1／n_2	IF Class is Numeric　(C_1)：Numeric
	スイッチの情報を調べセットされていれば（LA）を1に，セットされていなければ（LA）を0にする。スイッチは，パネルスイッチとユーザースイッチ（特定番地）の n_1 で示されるビット。
SWN　n_1	IF Switch On　(Pannel Sw. bit n_1)
SWF　n_1	IF Switch Off　(〃)
USN　n_1	IF Users Switch On　(Users Sw. bit n_1)
USF　n_1	IF Users Switch Off　(〃)
	C_1 で示される文字定数を n_3 桁並べたものと，C_2 から始まる n_3 桁とを比較して下記の条件を満足すれば（LA）を1に，満足しなければ（LA）を0にする。
FGC　C_1／C_2／n_3	Figurative Constant Greater　$(C_1) \geq (C_2)$
FGN　C_1／C_2／n_3	Figurative Constant Not Greater　$(C_1) < (C_2)$
FEC　C_1／C_2／n_3	Figurative Constant Equal　$(C_1) = (C_2)$
FEN　C_1／C_2／n_3	Figurative Constant Not Equal　$(C_1) \neq (C_2)$
FLC　C_1／C_2／n_3	Figurative Constant Less　$(C_1) < (C_2)$
FLN　C_1／C_2／n_3	Figurative Constant Not Less　$(C_1) \geq (C_2)$

命　令　形　式	機　　能
GOC　n_1 / n_2	Go To Depending On 次にn_2 個別記されているProcedure Name アドレスの中のn_1 番目のものを実行する。
TIM　n_1 / W_2	Time Zero Test $n_1 = 0$ならばスキップする。$n_1 \neq 0$ならばn_1 を1小さくしてW_2 へ飛ぶ。
BRN　W_1	Branch 無条件にW_1 へ飛ぶ。
NOP	No Operation
RETURN　W_1	Return W_1 の語の内容が示すアドレスの次の位置へ飛ぶ。
LAL　W_1	Load Logical Accumulator W_1 で示す語の内容を（LA）に入れる。
SLA　W_1	Store Logical Accumulator （LA）の内容をW_1 で示す語に移す。
ORR　W_1	Merge （LA）とW_1 で示す語の内容の論理和を（LA）に残す。
AND　W_1	Extract （LA）とW_1 で示す語の内容の論理積を（LA）に残す。
NOT	Reverse Logical Accumulator （LA）の内容の真偽を逆にする。

命　令　形　式	機　　能
IFF W_1	IF Logical Accumulator False (LA)が0ならばW_1 へ飛び，0でなければ次へ進む。
IFE W_1	IF Logical Accumulator True (LA)が1ならばW_1 へ飛び，1でなければ次へ進む。
CDB $C_1/n_2/W_3$	Convert Decimal to Binary C_1 から始まるn_2 桁の10進数を2進に変換してW_3 が示す語に入れる。
CBD W_1/C_2	Convert Binary to Decimal W_1 にある2進数を10進に変換してC_2 から始まる4桁に入れる。
ADA W_1/n_2	Arrange Data Address W_1 の内容から1を引いてn_2 を掛けW_1 に入れる。 桁数の指標を作るのに用いる。
WTC W_1/n_2	Word to Character W_1 の内容とn_2 の積をW_1 に入れる。語数と桁数の変換に用いる。
CTW W_1	Character to Word W_1 が示す桁数を語数に変換してW_1 に入れる。
CGB W_1/W_2	Change Bagin W_1 で示されるワードアドレスがW_2 に入る。 W_2 の最初の内容がW_1 アドレスの格納場所に入る。

命　令　形　式	機　　能
CGE W_1/W_2	Change End W_1 とW_2 の語の内容が変換される。
CRX	Clear Index システム指標レジスタをクリア
ADX $C_1/n_2/n_3$	Add Index C_1 から始まるn_2 桁の添字を2進に変えて指標レジスタに加える。
MDX C_1/W_2	Modify Character Address アドレスC_1 を指標レジスタで修飾してW_2 に入れる。

第3章　マイクロプログラミングによるコンパイラ機能の構成(10)

3.1　緒　　言

コンパイルの過程においては，ソースプログラムの解析とオブジェクトプログラムの生成のために，記号の処理，番地情報の処理などの非数値計算処理が中心となる。これらの処理に対して効果的に使用される命令系は，オブジェクトプログラム実行時の数値計算を中心としたものとは本来異なつたものである。マイクロプログラムをプログラム制御で随時交換することが可能な計算機においては，コンパイル過程ではそれに適した命令系を数値計算用のものととり換えて用いて，コンパイル過程の記憶装置利用効率および処理時間効率を向上することが可能となる。本章ではこの観点からこの形式の計算機に適したコンパイラ機能の構成の方法と実用計算機での適用例，その効果について述べる。

3.2　コンパイラにおける動的な表の管理

コンパイル過程における処理の中で使用頻度の大きいものとして表およびスタック（ここではこれらをまとめて扱う方法をとるので以下，単に表という）の処理があげられる。通常の実用言語に対してコンパイラで使用する表の個数は20～30個程度となり，これらのおのおのに固定の領域を与えると，処理の途中でどれか1個の表の領域を使い切つたときそれ以上の処理の続行ができなくなり，システムの処理容量上の制限がきびしく現われる。

ここでは，個々の表の容量を固定せずデータの増加に従つて表を拡大する方法をとり，1個の大きな領域を多くの表で共用することによつて，メモリ領域の利用効率をあげ，システムの処理容量上の制限を緩和することを考える。この方法では表の大きさとメモリ割付けがプログラムの進行につれて変化するので，これを動的な表管理法と呼ぶ。ここで扱う個々の表としては索引型の表およびスタックとし，それらの要素を固定長または不

定長であるとする。

動的な表管理法として，大きさが増減する表に対して増加と減少の双方に追随してメモリ割付けを変更する2方向性のものと，表の大きさの増大により必要となつた場合にのみ割付けを変更する1方向性のものが考えられる。いずれも表割付け変動に伴つてデータ移動を要するため処理時間増大の欠点をもつが，両者の中では1方向性のほうがその程度は少ない。メモリ利用の有効さは2方向性のほうがよい。表位置を固定する方法ではi番目の表領域の大きさをA_i，i番目の表の大きさS_iとして

$$S_i > A_i$$

が$i = 1, 2, \cdots, N$のどれかについて成立したとき表領域溢れとなり処理不能となるのに対し，1方向性の動的管理では表領域全体の大きさをA，i番目の表の最大の大きさを$S_{i\max}$として

$$\sum_{i=1}^{N} (S_{i\max}) > A$$

のときに表領域溢れとなり，2方向性の場合は

$$\sum_{i=1}^{N} S_i > A$$

のときに溢れとなる。要素が減少する表が1個でもあれば，システムの処理容量上の制限は2方向性のほうがゆるくなる。

ここでは1方向性の方法をとる。

3.2.1 表および指標

表の個数をNとし，i番目の表をT_iで表わす。$i = 1, 2, \cdots, N$である。i番目の表のj番目の要素をx_jで表わす。i番目の表の要素の個数をM_iとすると$j = 1, 2, \cdots, M_i$である。i番目の表に対し次の5個の指標をおく。

(1) 基点（表の先頭位置）：b_i

(2) 端点（表の終端位置）：e_i

(3) 下点（不定長要素の表の最終要素の基点側に付加された区切り印

の位置）：l_i

(4) 要素の大きさ：S_i （固定長要素の表で使用）

(5) キイ情報の大きさ：k_i

3.2.2 表領域中での表の割付け

連続番地から成る表領域Tにおいて表T_iはiの順に低番地から高番地へ割付けられる。Tの基点をb_T，端点をe_Tとする。基点b_Tは表T_1の基点b_1と一致するが，それぞれの表は必ずしも互に隣接せず，表の間あるいはT_Nの上方に未使用領域を残すことがある（図3.1）。

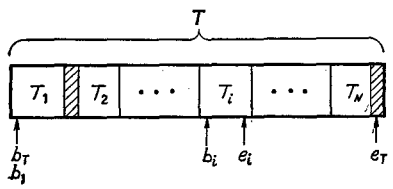


図3.1 表領域中での表の割付け

3.2.3 表中での表の要素の割付け

表T_iの領域内で要素x_jはjの順に低番地から高番地へ未使用領域を残すことなく割付けられる。要素x_1の最下点は表T_iの基点と一致し指標e_iで示される。

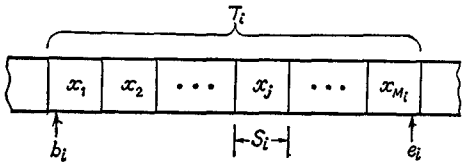
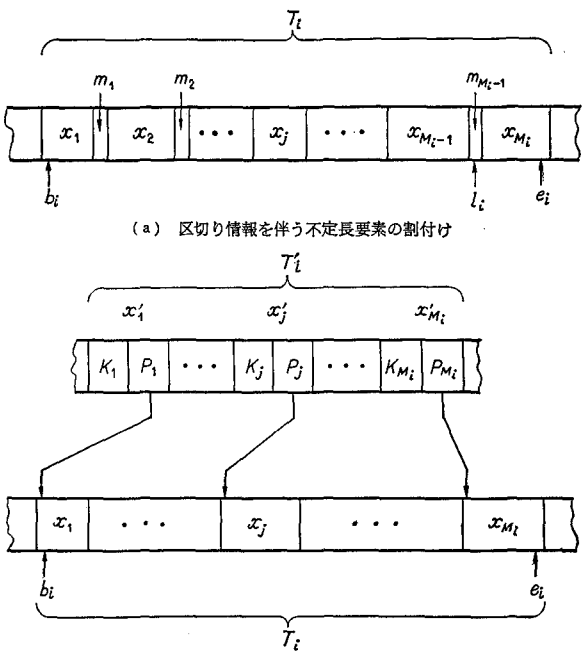


図3.2 固定長要素の割付け

要素x_jの大きさが固定されている固定長要素の表は図3.2のように構成される。

要素x_jの大きさが固定されていない表を不定長要素の表と呼ぶ。これには2種ある。一方は各要素間に区切り情報を挿入したものである。この形の表は不定長要素から成るスタックとして使用され，その端点での要素取出しに区切り情報を活用する。要素x_jとx_{j+1}の間の区切り情報をm_jとすると，表T_i中で最上位置にある区切り情報は要素x_{Mi}の下に付加されたx_{Mi-1}であり，この位置が表T_iの下点であつて指標l_iで示される〔図3.3(a)〕。もう一方は要素間に区切り情報を持たず，要素の大きさは要素中の特定部分に存在する情報で示される。各要素の位置を知るためにx_jの先頭位置を示す情報P_jを含む補

助表Ti'を使用する。補助表の構成は種々に考えられるが,ここでは固定長のキイK_jと指標P_jから成る固定長要素をもつとする。キイを固定長としたのは適当な内部形に変えられていることを想定するからである。補助表T_i'は前述の固定長要素の表の一つであり,その割付けは上述の規定に従う〔図 3.3(b)〕。



(a) 区切り情報を伴う不定長要素の割付け

(b) 補助表を用いる不定長要素の割付け

図 3.3

3.2.4 指標の取扱い

指標b_i, e_i, l_i, S_i, およびk_iは各表T_iについて1組あり，表領域中での各種の情報の位置を示すために使われる。これらの指標を表領域全体の分だけ集めて格納するために指標領域を設ける。

指標領域は$\boldsymbol{B}$, $\boldsymbol{E}$, $\boldsymbol{L}$, $\boldsymbol{S}$, および$\boldsymbol{K}$から成り，それぞれN個の要素から構成される。それぞれの中では指標が対応する表T_iのi番号順に低番地から高番地へ割付けられる。l_i, S_i, およびk_iの中には空となるものもあるが，表番号iから指標を引き出す過程を簡単にするために他の指標と同様の割付けを行なう。

3.2.5 表領域，指標領域，およびプログラム領域

表領域および指標領域は主メモリ中にあるとする。コンパイラはいくつかのフエイズに分けて構成されているのが普通であり，表領域および指標領域に置かれる情報はいくつかのフエイズにわたつて保持されなければならないので，これらの領域はプログラム領域とは区別して割付けられている必要がある。

3.2.6　表処理の初期条件

表領域の使用開始時点，たとえばあるプログラム単位のコンパイルの初めに，N個の表に対して指標b_i・e_i・l_iを全部e_Tに等しくする。これは，表への要素追加が進むにつれてメモリの高番地端から低番地の方向に表領域がひろがるような使い方を示している。指標S_iおよびk_iは表の種類と構造に応じてあらかじめ定まつている。

3.2.7　溢れの条件

プログラムの進行に伴つて表が拡大され，表領域の下端が低番地方向に伸びた結果，使用可能なメモリ領域を使い切つてみまつたときに，そのプログラムの処理続行が不可能となる。これはいわゆる表のパンクであり，表領域と他の領域との交さの形で現われる。

表領域の下端の内側にもう1個の表T_0を置き，e_0およびl_0の初期値をb_Tと等しくし，T_0だけは要素追加によつて高番地方向に伸びるような使い方をするならば，溢えの条件はT_0とT_1の交さ，すなわち$e_0 \geq b_1$で生じる。この方法では表領域そのものは固定されている（図3.4　）。

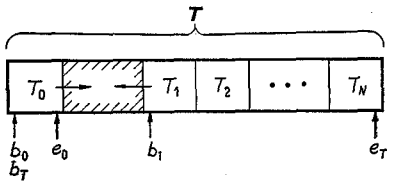


図3.4　表T_0を設ける場合の割付け

3.2.8　端点での要素追加

表の端点側を拡大して新しく要素を追加する。表の端点の上方に充分な空隙がなければ，表を下方に移動して空隙を作り出さなければならない。この表移動は動的な表管理の特徴的な処理であるが，これがプログラム処理時間増大の欠点となる。小きざみにひんぴんと表移動を行なうと時間増大が著しくなるので，移動幅を1回分の必要量より大きくとり表移動の頻度を減らして時間増大の軽減をはかる。この方式による端点での要素追加の機能は次のように表現できる。

(1)　$e_i + S_i < b_{i+1}$であれば(3)を行なう。

(2) $e_i + S_i \geq b_{i+1}$ であれば，T_1 から T_i までの表の全要素を下方に νS_i だけ移動し，それに伴い

$$\left.\begin{aligned} b_\lambda &\leftarrow b_\lambda - \nu S_i \\ e_\lambda &\leftarrow e_\lambda - \nu S_i \\ l_\lambda &\leftarrow l_\lambda - \nu S_i \end{aligned}\right\} \quad \lambda = 1, 2, \cdots, i$$

によつて指標を更新する。ν は移動幅を大きくとるための量で，正整数である。この過程で溢れの条件が成立すれば処理続行不能を表示する。

(3) $e_i + 1$ から上の領域に大きさ S_i の新しい要素を入れ，それに伴つて

$$e_i \leftarrow e_i + S_i$$

によつて指標を更新する。

不定長要素のスタックでの要素追加には 3.2.11 で述べる区切り設定をしたのち上記の方法を用いる。

3.2.9 表中での書込み，読出し

表の構造を変えることなくその中から情報を読み出し，また所定の位置に情報を書き込む操作は，動的な表管理とは本来無関係に必要な機能であるが，これらはこの体系では次のように行なわれる。

(1) 指標から実効番地を算出する。表中の要素位置は $b_i + (j-1) \times S_i$ で決まる。特に要素の中の語の相対位置 r を指定するときは $b_i + (j-1) \times S_i + r$ とする。

(2) 指定された位置の情報の出し入れを行なう。

3.2.10 端点での要素読出しおよび除去

スタックとして用いる表では端点にある要素の読出し，および除去が基本的な機能として必要である。

(1) 固定長要素の表の場合

端点要素の位置は $e_i - S_i + 1$ により指定できる。また要素除去に伴う指標更新は $e_i \leftarrow e_i - S_i$ により行なう。

(2) 不定長要素の表の場合

スタックとして用いる不定長要素の表は区切り情報をもつたものとし，指標 l_i をもとに区切り情報を見出して端点要素の位置を決める。要素除去に伴う指標更新は次のように行なう。

$$e_i \leftarrow l_i - 1$$

$$l_i \leftarrow b_i + l'$$

l'は端点要素下方の区切り情報の内容として保たれているもので，基点から端点の一つ手前の区切り情報までの距離を示している

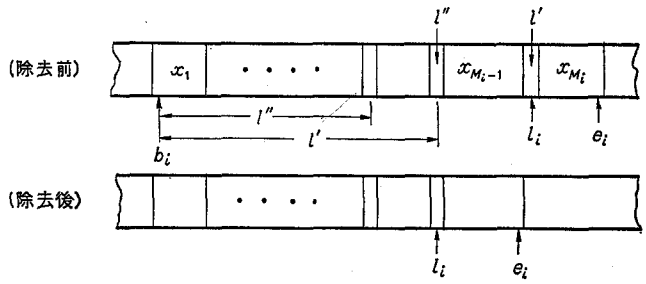


図 3.5 不定長要素の表の端点での要素除去

3. 2. 11 端点での区切り設定

これは不定長要素のスタックにおける要素追加に先立つて必要となる機能である。端点要素の上に区切り情報を積み，その内容として一つ下方の区切り情報の基点からの距離を入れる。この処理は次の操作で行なわれる。

$$e_i \leftarrow e_i + 1$$

$$l'_i \leftarrow l_i - b_i$$

l'_i は新しく設定された区切り情報の内容として入れる指標である（図 3.6 ）。この過程で表 T_i の大きさが

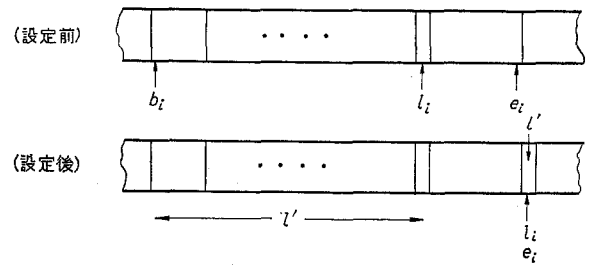


図 3.6 不定長要素の表の端点での区切り設定

増すから，これに先立つて空隙の存在を確認し必要に応じて表を移動させる操作を挿入しなければならない。これは 3. 2. 8 で述べたのと同様の方法で行なう。

この区切り設定の後で区切りの上に要素を積み上げる操作が伴うのが通例であるが，それには最初に述べた要素追加機能を使う。コンパイラではソース・プログラムの解析を進行させる過程で固定長要素の処理を基本としてスタックが作られ，ある段階に達するとそれまでに積み上げられた一群の要素を 1 個の不定長要素とみなす形の処理が多く，不定長の要素を直接に積上げ操作の対象とする必要はない。

3. 2. 12 表　引　き

(1) 固定長要素の場合

表引き処理は与えられたキイ情報（大きさ k_i ）と一致するキイをもつ要素（大きさ S_i，$S_i > k_i$）の位置 j を見出すことである。要素中のキイの位置はあらかじめ決められているとする。表引き処理は複合的な処理であり，この表管理の処理系を実現する命令系を活用して組み立てる。表引き実施範囲を指定するために指標 b_i，l_i e_i を使う。

(2) 不定長要素の表の場合

不定長要素の表の場合には固定長要素の補助表を使用する。まず上述の方法で補助表に対する表引きを行ない，そこから所要の指標 P_j を見出したうえ，不定長要素の表の要素を処理する（図 3.7 ）。

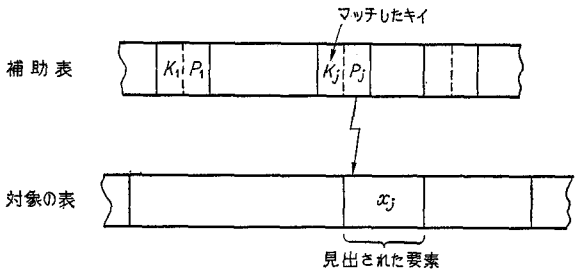


図 3.7　補助表を用いた表引き

3. 2. 13　スタックを利用したサブルーチン・ジャンプ

サブルーチン・ジャンプの際に復帰番地をスタックに積上げ，復帰に際しスタックを縮める方法をとるとサブルーチン・ネストの深さの制限を固定する必要がなくなる。サブルーチン・ジャンプに伴うパラメータの受渡しにもスタックを利用し，復帰に際して必要なだけ縮めることを可能にしておく。これらの操作のために復帰番地用と作業領域用の2種のスタックを使用するが，これらをあらかじめ特定番号の表の機能としておいて，サブルーチン・ジャンプ系のルーチン使用で自動的に選ばれて働くようシステム設計をすると用法が簡単になる。この形のサブルーチン・ジャンプを使用すると回帰的な呼出しの処理が可能となる。

3. 2. 14　スタックと待ち行列の変換

コンパイラで中間情報をスタックに積みながら処理を進め，ある区分まで構成できたときにそれをオブジェクト・プログラム列の原形としてスタックから逆順で取り出す操作が有効になる。この操作はスタックの待ち行列への変換としてとらえられるが，これを表処理の一つの機能としてまとめておく。スタックとして使う表の中で指定された部分の要素の並び方を反転させる操作を設けることでこれが行なわれ表からその部分を取り出すには通常の要素除去の機能を用いる。

3.3 COBOL コンパイラにおける実用例

前節に掲げた表管理の各機能は比較的小さな機能単位としてまとめられている。これらをマイクロプログラム・ルーチンとして実現すれば，それらを呼び出すことによつてそのような命令群があるかのような使い方が可能となる。すなわち，これらのコンパイラ機能がマイクロプログラム化されることになり，さらに，マイクロプログラムを交換することが可能な計算機においては，コンパイル時にこれらのコンパイル用マイクロプログラムをロードすることによつてコンパイルの効率を高めることが可能となる。

本節では，動的な表管理を交換可能なマイクロプログラムで実現した実用コンパイラの例として，MELCOM 3100 Mk−Ⅱ／Ⅲ COBOLコンパイラをとり，そのあらましを述べる。

3.3.1 COBOL コンパイラで用いる表の構造と用法

MELCOM 3100 Mk−Ⅱ／Ⅲ COBOL コンパイラでは合計３０個の表を動的な表管理の対象としている。これらの表の構造と種類および用法をまとめて表3.1に示す。

MELCOM 3100はキャラクタ単位にデータを処理する機能を基礎としているが，実際によく用いられる語長（１８ビット）単位の処理に対してマイクロ命令をもつているので，以下では情報の長さの単位としてこの語長を用いる。

表管理の基礎となる指標 b_i，e_i，および l_i は語単位で示されたメモリ番地に対応する。要素の大きさ S_i およびキイ情報の大きさ k_i も語を単位とするが，これらはそれぞれ5ビットおよび2ビットで表わし得るため，共に1語の中に入れ，制御語 C_i として用いる(図3.8)。全部の表についての C_i をたくわえるために領域 C を設けて S と K の代りに使用する。したがつて，指標領域は B，E，L，および C で構成される。

COBOL で使われる名前，予約語，その他プログラムに現われる語は，リテラル，編集用PICTURE指定およびラベル情報を除いてコン

パイラの字句解析段階で1語長の内部形に変換されている。そのためコンパイラの処理の対象となる情報はほとんど1語長以下の長さのものとして扱える。リテラルなど原形を保存する必要のあるものは不定長要素として扱い，補助表を伴う方法で処理を進める。表 3.1 で F/U の欄は固定長要素か不定長要素かを示している。特に F & U と書かれた場合は両方の扱いを受けることを示している。

表 3.1 MELCOM 3100 Mk-II/III COBOL コンパイラで使われる表

表の名前	*Si*	*ki*	*S/T*	*F/U*	機能・用途
MADLBL	1	1	*T*	*F*	前方参照 (Forward Reference) のアドレス生成のために用いる. READ 文の AT END 処理などで使用する.
AUX 1	1	1	*S*	*F* & *U*	condition の解析に使用. IF, PERFORM で現われる条件としてソース・プログラムで調べた情報を記憶する.
AUX 2	1	1			
AUX 3	1	1			
TEMP 1	1	1			算術演算の一時記憶に使用.
TEMP 2	1	1			複合条件の一時記憶に使用.
STCK 1	1	1			解析途上の一時記憶に使用. PERFORM～UNTIL～ PERFORM～AFTER～ IF 文 のように条件を伴う複雑な文のオブジェクト列の生成のために AUX 1～AUX 3 と組み合わせて使う.
STCK 2	1	1			
STCK 3	1	1			
MNT	1	1	*T*	*F*	DATA DIV. のファイル定義でレコード数チェックのための一時記憶として使用.
CORR 1	2	2	*S*	*F*	MOVE CORRESPONDING の送り出し側走査でレベル番号とデータ名を対にして記憶する. MOVE CORRESPONDIG の受入れ側走査で用いる.
CORR 2	2	2			
CTNT	1	1	*T*	*F*	紙テープコード変換表の ID 記憶用.
FNT	2	1	*T*	*F*	ENVIRONMENT DIV. の SELECT 句解析過程でファイル名と FDT ポインタが記憶され，DATA DIV. の FILE SECTION 解析過程で後者を NT ポインタに置き換える.
USET	3	2	*T*	*F*	USE 文の情報を記憶し，ラベル・ルーチン/エラー・ルーチンを作るファイルと出力時点をきめる.
SYM	2	2	*T*	*F*	SYMBOLIC KEY 項目名の記憶に使用.
ACT	2	2	*T*	*F*	ACTUAL KEY 項目名の記憶に使用.
SYSMAC	2	2	*T*	*F*	入出力処理で使用したシステム・マクロのシンボル名を登録し，オブジェクト列出力のときにマクロライブラリの ID 名に変換する.
REC	2	2	*T*	*F*	RECORD KEY 項目名の記憶に使用.
EXTNAM	2	2	*T*	*F*	プログラム・リンクのために使う外部名を記憶.
EXT	1	1	*S*	*F*	JSB ルーチンで使用，復帰アドレスを記憶する. EXT ルーチンで1要素が解放される.
WOT	1	1	*S*	*F&U*	表管理系内での一時記憶として種々の処理のために使用.
RST	2	1	*T*	*F*	手続き名の表．手続き名とその相対アドレスを積むフィールドとが対で記憶される.
GEST	4	1	*T*	*F*	データ階層の解析に使用.
GENROL	1	1	*S**	*F*	PROCEDURE DIV. のオブジェクト列を作って，随時出力する．*逆転操作の対象となる.
COT	1	1	*T*	*U*	ラベル情報，編集用 PICTURE 指定．条件名の VALUE, PROCEDURE LITERAL を入れる.
NT	2	1	*T*	*F*	呼び名，ファイル名，レコード名，データ名，条件名などの記憶．要素は2語長で，前半に名前の内部形，後半に詳細情報または他の表へのポインタがはいる.
RDT	1	1	*T*	*U*	レコード名，データ名，プロセデュア・リテラルの詳細情報の記憶．NT を補助表として参照する.
FDT	13(Mk-II) /21(Mk-III)	1	*T*	*F*	ファイルに関する情報の記憶．実行時用 FDT，およびマクロ命令生成のために使う.

3.3.2　表管理マイクロプログラム

MELCOM 3100 Mk−Ⅱ／Ⅲ COBOL コンパイラのために用意したマイクロプログラムの主要なものについて機能および処理内容，それらを呼び出す命令形（これがマクロ・アセンブラ言語機能として使われる）をまとめて表 3.2 に示す。操作内容を表現するために使用した記号は次のとおりである。

i　　表番号

$i = w$ のとき表 WOT を，$i =$ R のとき表 EXT を示し，E_i の代りに E_w, E_R と書く。

j　　表中の要素番号

P_{ij}　　表要素 x_{ij} を指す指標

$L.A.$　　論理アキュムレータ

(A)　　番地 A の内容

$((A))$　　番地 A の内容が示す番地の内容

$(A) \rightarrow (B)$　番地 A の内容を番地 B へ転送。

$(A_1 \sim A_2) \rightarrow (B_1 \sim B_2)$　番地 A_1 から A_2 までの連続領域の内容を番地 B_1 から B_2 までの連続領域へ転送。

$(C_i) \Longrightarrow i, j$　制御語 C_i から i と j とを抽出。

$P_{ij} \Longleftarrow \langle i, j \rangle$　i と j とから指標 P_{ij} を合成。

Search T_i〔$A_1 \sim A_2$, k_i, S_i〕　表 T_i を表引き操作。範囲は番地 A_1 から A_2 まで。

Flip T_i〔$A_1 \sim A_2$〕　表 T_i の番地 A_1 から A_2 までの内容を語単位で順序逆転。

Perform α　ラベル α のついた手続き（STRL 命令欄に記載）を行なう。

表 3.2 MELCOM 3100 Mk-II/III COBOL コンパイラの表管理マイクロ・プログラム・ルーチン機能一覧

命令形	機能	操作内容	備考
STRL w_1/w_2	端点に1要素 (S_i) 追加	$(w_1)\Rightarrow i, j$; $(C_i)\Rightarrow S_i$; $(E_i)\rightarrow(E)$; $S_i\rightarrow(S)$; $(B_{i+1})\rightarrow(B)$; perform α; $(w_2\sim w_2+S_i)\rightarrow((E_i)+1\sim E_i+1+S_i)$; $(E_i)+S_i\rightarrow(E_i)$; α: if$(E)+(S)<(B)$ then return else if $(B_1)-\nu\times(S)\leq(E_0)$ then O'FLOW EXIT else $((B_1)\sim(E))\rightarrow((B_1)-\nu\times(S)\sim(E)-\nu\times(S))$	*
STRR w_1/w_2	表中に1要素 (S_i) 書込み	$(w_1)\Rightarrow i, j$; $(C_i)\Rightarrow S_i$; $(B_i)+j\times S_i\rightarrow(X)$; $(w_2\sim w_2+S_i)\rightarrow((X)\sim(X)+S_i)$;	*
STO $w_1/w_2/n_3$	表中に1語書込み	$(w_1)\Rightarrow i, j$; $(C_i)\Rightarrow Si$; $(B_{ij}+j\times S_i+n_3\rightarrow(X)$; $(w_2)\rightarrow((X))$;	
LDR w_1/w_2	表中から1要素(S_i)読出し	$(w_1)\Rightarrow i, j$; $(C_i)\Rightarrow S_i$; $(B_i)+j\times S_i\rightarrow(X)$; $((X)\sim(X)+S_i)\rightarrow(w_2\sim w_2+S_i)$;	*
LDO $w_1/w_2/w_3$	表中から1語読出し	$(w_1)\Rightarrow i, j$; $(C_i)\Rightarrow S_i$; $(B_i)+j\times S_i+n_3\rightarrow(X)$; $((X))\rightarrow(w_2)$	
PNG w_1/w_2	ポインタ合成	$(w_1)\Rightarrow i$; $(C_i)\Rightarrow S_i$; $j=\left[\frac{(E_i)+1-(B_i)}{S_i}\right]$; $P_ij\Leftarrow\langle i, j\rangle$; $P_ij\rightarrow(w_2)$;	
CPE w_1/w_2	ポインタからアドレス抽出	$(w_1)\Rightarrow i, j$; $(C_i)\Rightarrow S_i$; $(B_i)+j\times S_i\rightarrow(w_2)$	
SRA $w_1/w_2/w_3/w_4$	表引き $(w_1)=i$ $(w_2)=$キイ $(w_4)=$ジャンプ先	$(w_1)\Rightarrow i$; $(C_i)\Rightarrow ki, S_i$; Search T_i $[(L_i)+1\sim(E_i)$; $ki, S_i]$ if match at j then $P_ij\Leftarrow\langle i, j\rangle$; $P_ij\rightarrow(w_3)$; goto w_4; else goto next;	
SRB $w_1/w_2/w_3/w_4$	表引き $(w_1)=P_ij$ $(w_2)=$キイ $(w_4)=$ジャンプ先	$(w_1)\Rightarrow i, j$; $(C_i)\Rightarrow ki, S_i$; $(B_i)+j\times S_i\rightarrow(X)$; Search $T_i[(X)\sim(E_i); ki, S_i]$ if match at j_1 then $P_ij\Leftarrow\langle i, j_1\rangle$; $Pij\rightarrow(w_3)$; goto w_4; else goto next;	
RSV w_1	端点に区切り情報を設定	$(w_1)\Rightarrow i$; $(E_i)\rightarrow(E)$; $1\rightarrow(S)$; $(B_{i+1})\rightarrow(B)$; perform α; $(E_i)+1\rightarrow(E_i)$; $(L_i)-(B_i)\rightarrow((E_i))$; $(E_i)\rightarrow(L_i)$;	α は STRL 参照
REL w_1	端点から1要素と1区切り情報を除去	$(w_1)\Rightarrow i$; if $(L_i)=(B_i)$ then $(B_i)\rightarrow(E_i)$; else if $(L_i)>(B_i)$ then $(L_i)-1\rightarrow(E_i)$; $(B_i)+((L_i))\rightarrow(L_i)$;	
MOA w_1/w_2	端点から1語取出し	$(w_1)\Rightarrow i$; $((E_i))\rightarrow(w_2)$; $(E_i)-1\rightarrow(E_i)$; if $(E_i)=(L_i)$ then $0\rightarrow(L.A.)$ else $1\rightarrow(L.A.)$	
CAR w_1/w_2	端点1要素を他に移転 $(w_1)=i$ $(w_2)=i_1$	$(w_1)\Rightarrow i$; $(E_{i1})\rightarrow(E)$; $(E_i)-(L_i)\rightarrow(S)$; $(B_{i+1})\rightarrow(B)$; perform α; $((L_i)+1\sim(E_i))\rightarrow((E_{i1})+1\sim(E_i)-(L_i)+1)$; perform REL;	α は STRL 参照
REF w_1	端点から1要素除去	$(w_1)\Rightarrow i$; $(C_i)\Rightarrow S_i$; $(E_i)-S_i\rightarrow(E_i)$;	
REP w_1/w_2	端点から1要素除去	$(w_1)\Rightarrow i$; $(C_i)\Rightarrow S_i$; $(E_i)-S_i\rightarrow(E_i)$; if $(E_i)=(B_i)$ then goto w_2 else goto next;	
FET w_1	端点に1語追加	$(E_w)\rightarrow(E)$; $1\rightarrow(S)$; $(B_{w+1})\rightarrow(B)$; perform α; $(E_w)+1\rightarrow(E_w)$; $(w_1)\rightarrow((E_w))$;	α は STRL 参照
ASP w_1	端点から1語取出し	$((E_w))\rightarrow(w_1)$; $(E_w)-1\rightarrow(E_w)$;	
ASK w_1	端点で1語読出し	$((E_w))\rightarrow(w_1)$;	
POW n_1	端点から n_1 語除去	$(E_w)-n_1\rightarrow(E_w)$;	
JSB	サブルーチン・ジャンプ	$(E_R)\rightarrow(E)$; $1\rightarrow(S)$; $(B_{R+1})\rightarrow(B)$; perform α; $(E_R)+1\rightarrow(E_R)$; 復帰アドレス$\rightarrow((E_R))$; goto w_1;	α は STRL 参照
EXN	復帰 $((E_R))=$復帰アドレス	$(E_R)\rightarrow(X)$; $(E_R)-1\rightarrow(E_R)$; goto (X);	
EXF	復帰 $((E_R))=$復帰アドレス	$0\rightarrow(L.A.)$; $(E_R)\rightarrow(X)$; $(E_R)-1\rightarrow(E_R)$; goto (X);	
EXT	復帰 $((E_R))=$復帰アドレス	$1\rightarrow(L.A.)$; $(E_R)\rightarrow(X)$; $(E_R)-1\rightarrow(E_R)$; goto (X);	
JAF w_1	論理分岐	if $(L.A.)=0$ then goto w_1;	
JAT w_1	論理分岐	if $(L.A.)=1$ then goto w_1;	
FLP	端点要素を反転	$(w_1)\Rightarrow i$; Flip $T_i[(L_i)+1\sim(E_i)]$	

* 特定の表 (RDT) については要素転送の過程で，パッキングまたはアンパッキングが行なわれる．

3.3.3　COBOLコンパイラでの表処理の例

上記のCOBOLコンパイラでは随所に動的な表処理をしているが，その効果的な使用例として，PROCEDURE DIVISIONの解析とオブジエクトプログラム列の生成を受けもつフエイズ5でのGENROL表の操作をとりあげる。その概要は図3.9に示すとおりである。

フエイズ5への入力はタイプ3と呼ばれる中間情報で，これはソースプログラムに字句解析を施したのち，1語長単位のコード化した内部形に変え，さらにそれ以後の解析に適するように整理したものである。

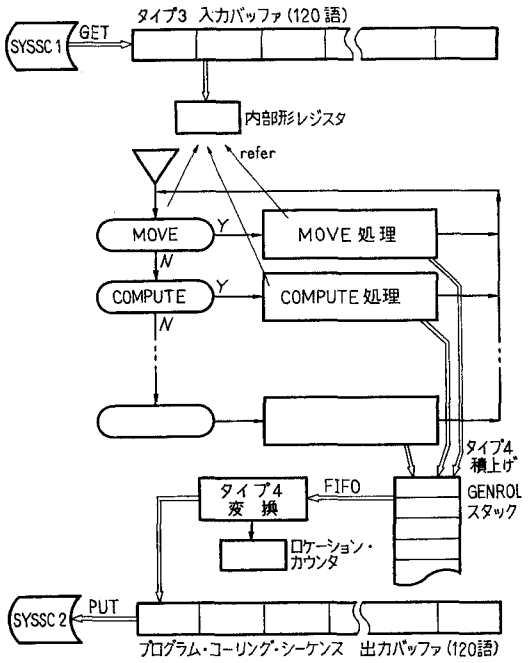


図3.9　COBOLフェイズ5でのPROCEDURE DIVISIONの処理

入力バッファに入つたタイプ3の情報は語単位で走査される。これはコンパイラが持つている文法原形表とのつき合わせをもとにした方法で行なわれる。[11] ここではPROCEDURE DIVISIONを構成する交の動詞を識別し，その文の処理ルーチンで解析を行なつたのち,再び次の動詞の識別に進む。この走査はタイプ3のエンドマークが現われるまで続く。それぞれの交の解析では，タイプ3を1語ずつ読み進めながら構文の適正さを調べ，オブジエクトプログラム原形の列を生成しGENROL表に積み上げて行く。IF文，READ文のAT END指定，および算術文のSIZE ERROR指定のように文の中に文を含む場合には，動詞を識別してその処理ルーチンへ飛ぶためのルーチンを回帰的に呼び出す。これは前述のJSB命令の復帰番地をスタックする機能により実現される。またそれぞれの文の処理ルーチ

ン中での複雑なサブルーチンのネストに対しても，JSB命令とEXT命令が有効に使われている。

GENROL 表にスタックされたオブジェクト原形列は，タイプ3の1語読出しルーチンでソースプログラムの行（カード1枚）の区切りのマークを検出した時点に，FLP命令による要素の逆転を施したのち取り出される。ELP命令はGENROLの内容をFIFO（first-in first-out)で処理するために使用される。GENROL の内容は1語ずつ取り出され，オブジエクトプログラムを構成する語が現われると相対番地を教えるロケーション・カウンタを1加算し，前方参照のフラグが現われるとMDLBL に，パラグラフかセクションのコードが現われると，PSTにそのときのロケーション・カウンタに値を入れる。こうして，次のフエイズで機械的に指標をそれが示す情報で置き換えれば，オブジエクトプログラム列になるようにしたのち，タイプ4として出力される。

3.3.4 動的な表管理の効果

動的な表管理のねらいは初めに述べたように，表領域の有効利用による処理容量制限の緩和にある。これの具体的な効果として，MELCOM 3100 MK-Ⅱ/Ⅲ COBOL コンパイラでは，ソースプログラムを書く上で課せられる制限事項の多くを表面に固定的に出さなくすることができた。このコンパイラの前の版であるMK-I COBOL コンパイラからの制限事項の緩和の状況は表3.3に示す通りで，表中11項目のうち3項目を除いて動的管理の対象として，固定的な制限がはずされた。

また，固定的な制限の場

表3.3 COBOL コンパイラにおける表の制限事項

制限項目	Mk-I COBOL コンパイラ	Mk-II/III COBOL コンパイラ
異なるデータ名の文字総数	24,000	24,000
異なるリテラルの文字総数	1,500	1,500
セクション名，パラグラフ名総数	200	300
データ名，FILLER，PROCEDURE DIV. 中の異なる値のコンスタントの個数，ENVIRONMENT DIV. の SPECIAL NAMES 中の名前の総和	700	—
ファイル数	10	—
ファイル名の2倍と FILE SECTION 中のレコード名との和	50	—
レベル番号チェックレベルの深さ	70	—
修飾語	6	—
ADD, SUBTRACT における加数，減数の個数	9	—
IF 命令の個数	200	—
GO TO P_1 P_2…P_n DEPENDING 命令の P_i の個数 n	30	—

合は，それぞれの表ごとに余裕をとるが，動的管理の場合はその必要がなく，表領域をきりつめ得るのでプログラム領域を大きくでき，その結果，コンパイラを構成するフエイズ数の減少も可能となる。これはコンパイル時間の短縮につながり，動的な表管理の副産物的効果の一つとなる。

その他の効果としては，表の管理を統一的に行なえるため，表処理のプログラミングに伴う指標処理のわずらわしさを除けること，サブルーチンの回帰呼出しが可能となることがあげられる。

動的表管理の欠点としては処理時間の増大がある。これについては動的表管理の有無だけを取り出した比較は行なつていないが，中間言語としてアセンブラ語をとる間接コンパイル方式で，かつ動的な表管理を用いていないMK－I COBOLコンパイラから，直接コンパイル方式でかつ動的な表管理を採用したMK－Ⅱ／Ⅲ COBOL コンパイラになつて，コンパイル速度が約4倍に改善された実績をもつている。これは動的な表管理のマイナス効果を含み，さらにMK－Ⅱ／Ⅲ コンパイラではリンク・エデイト時間をも含めての比較であるので，実用的には動的な表管理の採用によつて性能がおちたとはいえず，全体としての向上の方が著しい。MK－Ⅱ／Ⅲ COBOL コンパイラによるコンパイル所要時間は，リンク・エデイトを含めて平均毎分約１５０行となつている。

3.4 その他の機能

本章では，コンパイラにおいて特に通常の処理と異つた処理として，特徴的な表の管理とそれに関連する機能を中心として議論を進めて来た。前節までに記述したものが，表管理を目的としたマイクロプログラムである。コンパイラで行なわれる処理は，表管理のほかにも種々特徴的なものがある。それらについてもよく吟味して，一般のマイクロプログラムを用いるよりも効率を高めるような特殊なマイクロプログラム系を構成することが可能であろうし，それに成功すればコンパイラの構造を簡単にして行くこ

とが可能であろう。しかしMELCOM−3100においては，前述のもの以外に組織だつてマイクロプログラムを作ることはしなかつた。MELCOM−1530において最初に開発された解析法，生成法がその後の文法仕様の拡大に対し充分な融通性を示し，基本マイクロプログラムのみで足りたからである。

しかし，表管理のために新しいマイクロプログラム系を導入した機会に，一般用のマイクロプログラムの幾つかに対して手を加えた。これらはコンパイラ構成のためのマイクロプログラムという形でとらえ得るので，ここにあげる。

これらはもともと一般用のマイクロプログラムとして備えられているもののうち，特にコンパイラでよく使用するものについて，命令の形式を単純化し，従つてバリエーションをなくして専用化したものが主である。そのやり方を延長して，簡単な形の専用命令として新設されたものも幾つかある。これらのマイクロプログラムの命令形式と機能をまとめて表3.4に示す。

表 3.4

命令形式	機能
BLC W_1／W_2／BA	(W_1)＝(W_2) ならばBAへ飛ぶ。 W_1，W_2 はワードアドレス。
BLU W_1／W_2／BA	(W_1)≒(W_2) ならばBAへ飛ぶ。 W_1，W_2 はワードアドレス。
BAC n_1／n_2／BA	アキュムレータの内容 x が $n_1 \leq x \leq n_2$ ならBAへ飛ぶ。
BBF n_1／W_2／BA	W_2 の n_1 ビット目が0ならBAへ飛ぶ。
SME W_1／W_2／BA	{(W_1)AND(W_2)}＝(W_2+1)ならばBAへ飛ぶ。
MEM W_1／W_2	(W_1)OR(W_2)をW_2へ入れる。
SMX W_1／W_2	(W_1)AND(W_2)をW_2へ入れる。
CLZ W	Wの内容を0にする。
STM W	Wの内容を−1にする。

たとえばBLCという命令はBMEという一般用の命令がオペランドのデータ形式に自由度が多く，その指定と判定の操作が伴う点を改め，オペランドのデータ形を固定したものである。これによりプログラム作成の単純化と処理効率の向上を狙っている。

これらのマイクロプログラムは，一般用のものと全く別個に作る必要はなく，一般用のマイクロプログラムを多入口点化し若干の修正を施すことにより，実現できるものが多いから，マイクロプログラム用記憶領域を余り増大させることなく実装することができた。

3.5 結　　言

マイクロプログラムをプログラム制御で随時交換することが，可能な計算機の特徴を生かし，非数値計算処理が中心となるコンパイル過程に適した命令系を用いて，コンパイル過程の記憶装置利用効率および処理時間効率を向上する観点から，コンパイラに多用される表の管理を動点に行なう一連の処理体系と，COBOLコンパイラにおける実用例について述べた。例に用いたCOBOLコンパイラは初版完成以来基本的な改造なく，現在も第一線で稼動しているものである。

第4章　結　　論

本編ではマイクロプログラミング技術を活用したコンパイラの構成を主題とし，特に，マイクロプログラムの交換によつて，オブジエクトプログラムの基本機能のマイクロプログラム化による実行時の処理系，およびコンパイラの基本機能のマイクロプログラム化によるコンパイル時の処理系をそれぞれの処理に適合した形で実行する方法を述べた。具体的な実施例としてCOBOLコンパイラの実行時処理系およびコンパイル時処理系をとり上げ，その主要な機能の構成について述べた。

実行時処理系すなわちオブジエクト計算機をマイクロプログラムで構成することにより，計算機がもつ構造と大幅に異つた構造をもつたソース言語に対して適合しやすい処理機能を形成できるので，コンパイラにかゝる負担を相当軽減できる。これにより，制御用計算機に近い構造をもつ計算機において，その規模に対してやゝ豊富に過ぎるオプションをもつCOBOLコンパイラをかなり少い労力で実現することができた。

さらにこのマイクロプログラム・ルーチンをそのまゝ活用して，事務用の問題向き言語RPGの一種であるACEコンパイラのオブジエクト計算機として用い，COBOLと同様の効率を上げることができた。

コンパイル時機能については，代表的なものとして表およびスタックの処理をとり上げ，多くの表に対して1個の共通メモリ領域でその割付けと各種の処理に伴う移動を統一的に行なう動的な表管理法とその実用例を述べた。この方法によつてコンパイラの処理容量の制限条件が緩和され，各種の形態のプログラムに対して常に共通メモリ領域全体の溢れに至るまでの処理能力を発揮することが可能となつたが，この機能を効率よく実現する上でマイクロプログラミングの活用が寄与している。

このような交換可能なマイクロプログラムは最近診断用プログラムに適用される傾向があるが[9]，ここでは計算機機能そのものの実現に用いている。

なお，これらのマイクロプログラム・ルーチンは，オペレーテイングシステ

ムの制御によつてオブジェクトプログラムと同時に，あるいはコンパイラのシステムローディングの過程でメモリ内に装荷される。従つてプログラマあるいはオペレータが手操作を介入する必要はない。

このようなマイクロプログラムによる計算機機能の実現により，各機能の関係が体系的に明確化されるため，直接機械語を用いてコンパイラと実行時ライブラリを作成した場合に比べてプログラムのモジュラリティが良く，言語機能の拡張などに伴うプログラム修正，保守が容易であると報告されている。反面，ソフトウエアを供給する立場では，マイクロプログラム・ライブラリの管理という作業が発生することになり，これがこの方式のデメリットといわれるが，実際にはこの方式によるコンパイラの簡単化の効果が大きいため，作業量としては差引充分な効果を上げている。

〔第Ⅰ編　参考文献〕

(1) Wilkes,M.V., Renwick,W., and Weeler,D.：〝The Design of a Control Unit of an Electronic Digital Computer,〟Proc. IEEE,105,P.121(1958)

(2) 高橋，三上：〝MELCOM−1530 データプロセッシングシステム── そのストアドロジック設計，〟三菱電機技報，38−8，P.61(昭39−08)

(3) 首藤，関本，魚田，鶴岡：〝MELCOM−1530 COBOL システムの構成，〟昭39信学全大，554(昭39)

(4) 首藤，関本，鶴岡：〝MELCOM−1530 COBOLシステムに於ける言語変換機構とオブジエクトの構成，〟昭40電四連大，638(昭40)

(5) Melbourne, A.J., and Pugmire, J.M.：〝A Small Computer for the Direct Processing of Fortran Statements,〟Computer J. 8, P.24(1965)

(6) Bashkow, T.R., Sasson, A., and Kronfeld,A.：〝System Design of a FORTRAN Machine〟IEEE Trans., EC−16−4, P.485 (1967)

(7) 情報処理学会COBOL研究会：〝国産のCOBOLコンパイラ,〟情報処理，8−3，P.140(昭42−05)

(8) 国分，有坂，首藤，魚田：〝MELCOM−3100 ソフトウエア(5) ── ACE コンパイラシステムの概要──,〟三菱電機技報，42−10，P.1360 (昭43−10)

(9) Husson, S.S.：〝Microprogramming,Principles and Practices,〟Prentice−Hall(1970)

(10) 首藤，小碇：〝動的な表管理とCOBOLコンパイラへのその応用〟情報処理，13−2，P.113(昭47−02)

(11) 日本電子工業振興協会：〝MELCOM 1530 COBOL(コンパイラ編)〟，(1966)

⑫ 三菱電機(株)：〝MELCOM COBOL 説明書 G1－TLO1－00A ＜7004＞,〃(1970)

⑬ 三菱電機(株)：〝MELCOM 3100 MARK－Ⅲ COBOL説明書 G4－TLO1－00A＜7001＞,〃(1970)

⑭ 三井，他：〝MELCOM－3100デイスクオペレーテイングシステム(1)〃, 三菱電機技報，43－11，P.1539(1969)

第Ⅱ編　コンパイラ生成過程の自動化

第１章　緒　　論

電子計算機の利用法が高度化し，多岐にわたるにつれてソフトウエアでカバーする領分が増加し，ソフトウエア・コストの割合が次第に増している。一つの応用システムを実動させるに要するプログラムの量，あるいは一つの新機種を開発するに伴つて必要となるソフトウエアの量は，ぼう大なものとなり，従来の様に手作業で多くの人手と時間をかける方法では，システムの開発速度の要求に応えられなくなりつゝある。そのため，ソフトウエアの作成過程を自動化し，効率を上げる必要が大きい。特に新しい機種を採用あるいは開発する場合には，ソフトウエアの整備は乗り越えねばならない大きな問題となる。

応用プログラムについては，FORTRAN, COBOLなどの標準言語を用いていれば，機種移行に際しても比較的容易に対処できるようになつてきたが，オペレーテイング・システム，コンパイラなどのシステムプログラムについては，まだこの問題が残つている。機械の方式に大きく依存するオペレーテイング・システムの基本的な部分は困難が大きいが，比較的機械依存度の低いコンパイラ類については，機種移行に対する作業能率向上の手段が見出し得ると考えられる。

コンパイラの論理を標準手続き向き言語で書こうとする試みは，従来から行なわれているが，FORTRAN, ALGOL などを用いると処理対象のデータ型，演算機能で点で無理が大きく，種々の補助手段を用いたとしても冗長なプログラムを要することになり，現実的ではない。PL／Ｉを用いた試みも報告されており，この場合は，言語が豊富な機能をもつているので一応書き切つている。しかしPL／Ｉはその機能豊富さを裏付けるために多量の実行時プログラムを要すること，またPL／Ｉのステートメントそのものが，システムプログラム向きに重点を置いて作られていないことなどから，実行効率の上で問題があり，むしろこの場合の最大の効果はシステムプログラムの論理をかなり機械無依存な

方法で表現したことにあるといえよう。

システム記述言語をとりあげる場合，その処理対象を定式化して，それに対する記述言語を設計するという技術的な問題の他に，その記述言語を実用するためのプロセッサの開発と，それ以後の適用による効果とのコストバランスの問題があり，全体としてソフトウエア・コンパテイビリテイが重要視される機種開発環境にあるという事情が，システム記述言語の共通化普遍化の妨げとなつている。

ここでは，一つの試みとして機械的手続きで作成するという観点から，コンパイラの構成を考察しておき，ある言語に対するコンパイラのひな形をまず手書きで作成して，異つた計算機に適用するためにコンパイラのうち，計算機ごとに異なる部分を各計算機の特性に応じて，置き換えて所要のコンパイラを得る方法を考える。

第2章　コンパイラ生成過程の自動化(7)

2.1　緒　　言

計算機ソフトウエア開発のために計算機を活用することの一つとして，新機種開発時にすみやかにソフトウエアをそろえる要求に応じるため，コンパイラを作成する過程に機械処理を導入する方法について述べる。前章で触れたように，コンパイラの一部分を計算機の特性に合わせて，置きかえることを中心としており，置換形によるコンパイラ生成自動化と呼ぶべき方法である。この過程を計算機に実行させた結果を報告する。

2.2 置換法によるコンパイラ自動作成

計算機X用のコンパイラを計算機Xを用いて計算機Y用のコンパイラに変換することを問題とする。

一般にコンパイラは，ソース言語L_a，オブジェクト言語L_b，およびコンパイラ自身が書かれている言語L_cを与えて，

$$C\begin{matrix}L_a \rightarrow L_b \\ L_c\end{matrix} \qquad (1)$$

のような形で表わすことができる。しかし異なつた機械の間でコンパイラの写しかえをする場合には，コンパイラに関係する言語の他に，その言語を計算機内部にもち込む文字コードについても明確に規定しておかねばならない。そこで記法(1)にコードを表わす添字を追加し，記法(2)のように書く。

$$C\begin{matrix}L_{a,x} \rightarrow L_{b,y} \\ L_{c,x}\end{matrix} \qquad (2)$$

これは「コードxで表現された言語L_aを，コードyで表現された言語L_bに変換する機能をもち，自身はコードzで言語L_cを用いて書かれているコンパイラ」を表わしている。

置換法によるコンパイラ作成のためには，種となるコンパイラが必要である。種となるコンパイラは計算機Xで稼動できるものでなければならないが，置換のことを考慮してソース言語L_sを用いて書いておく。オブジェクト言語はL_xであるから，このコンパイラは記法(3)で表わされる。

$$C\begin{matrix}L_{s,x} \rightarrow L_{x,x} \\ L_s\end{matrix} \qquad (3)$$

ただし，コンパイラが書かれている言語L_sが第2添字をもたないのは，コーディングシート上に書かれていて計算機にインプットされるものでないことを表わしている。

この段階ではL_sで書かれたプログラムはまだ計算機Xで実行可能では

ないので，このコンパイラもまた実行可能ではない。

しかし，コンパイラのプログラム論理はL_sを用いて書かれているから，これをマクロ命令構成法などを用いて機械的にL_xの命令群に書きかえることによつて計算機Xで稼動するコンパイラ，

$$C\begin{matrix}L_s,x\rightarrow L_x,x\\L_x,x\end{matrix} \tag{4}$$

を容易に得ることができる。

目標は計算機Yで稼動するコンパイラ

$$C\begin{matrix}L_s,y\rightarrow L_y,y\\L_y,y\end{matrix} \tag{5}$$

である。コンパイラ(3)は計算機Xの基本的な機能に関する記述がもり込まれているからこの部分を計算機Y用に書きかえることによつてL_sをL_yに変えるコンパイラ

$$C\begin{matrix}L_s,y\rightarrow L_y,y\\L_s\end{matrix} \tag{6}$$

が得られる。これは所要コンパイラと同じ機能をもつているが，書かれている言語がL_sであるからこのまゝでは稼動不能である。これを$L_y,_y$にするためにはコンパイラ(6)を計算機Yのコードで表現した

$$C\begin{matrix}L_s,y\rightarrow L_y,y\\L_s,y\end{matrix} \tag{7}$$

を

$$C\begin{matrix}L_s,y\rightarrow L_y,y\\L_x,x\end{matrix} \tag{8}$$

によつて計算機Xを用いて処理すればよい。この場合の道具となるコンパイラ(8)は，コンパイラ(4)が読みとるコードをxからyに修正して作つた

$$C^{L_s,y\to L_x,x}_{L_x,x} \qquad (9)$$

を用いてコンパイラ(7)を処理することによつて得られる。ところが，計算機Xの入力装置はコードxでしか読みとることができないから，コンパイラ(7)はコードxで表現されたコンパイラ

$$C^{L_s,y\to L_y,y}_{L_s,x} \qquad (10)$$

の形でなければならない。一方，道具となるコンパイラは所要コンパイラからの要請によつてすべてコードyでソースプログラムを読みとるようになつているから，両者をつなぐコード変換機構が必要である。

以上の手順を処理順序に従つて図示すると図2.1のようになる。この図で2ケ所に出ている記番⊗はxからyへのコード変換を意味している。

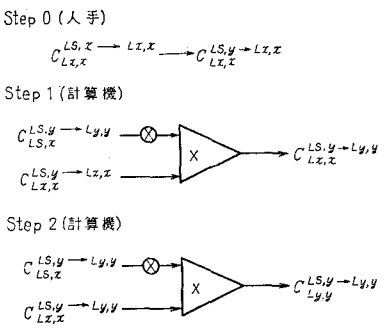


図2.1　計算機Xによる計算機Yのためのコンパイラの作成（コード変換機構を用いる方法）

さて以上は，所要コンパイラを

$$C^{L_s,y\to L_y,y}_{L_y,y}$$

として議論を進めてきたのであるが，それには図2.1のStop 0のコードの置き換えが簡単に行ない得ることが条件となつていた。この条件は一連の置換手順の結果得られるコンパイラの入力コードや機械記述が計算機Xのものであつても事後に置き換え処理が可能であるということになり得る。この考え方で，計算機を用いて行なう置換処理によつて得られるコンパイラを

$$C^{L_s,x\to L_y,y}_{L_y,y}$$

と仮定すれば，図 2.1 の系は図 2.2 に示すような系になる。図 2.1 と比較すると入力コードなどの置きかえが最後にきており，しかも機械手順におけるコード変換機構が不要になつている。

以上の過程を通じて人手の介在を必換とする個所は，

(1) 種コンパイラのソース言語版を所要コンパイラのソース言語版に書きかえる作業，すなわち，

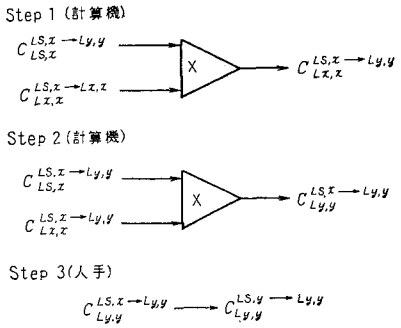


図 2.2 計算機Xによる計算機Yのためのコンパイラの作成（コード変換機構を用いない方法）

$$C^{L_s,x\to L_x,x}_{L_s} \Rightarrow C^{L_s,y\to L_y,y}_{L_s}$$

または

$$C^{L_s,x\to L_y,y}_{L_s}$$

(2) 種コンパイラまたは置換されたコンパイラの入力コードや機械記述を計算機Y用に置きかえる作業，すなわち，

$$C^{L_s,x\to L_x,x}_{L_x,x} \Rightarrow C^{L_s,y\to L_x,x}_{L_x,x}$$

または

$$C^{L_s,x\to L_y,y}_{L_y,y} \Rightarrow C^{L_s,y\to L_y,y}_{L_y,y}$$

である。この2つの操作を簡単に行なえるか否かが置換法の成否を左右する重要なポイントである。そのために，コンパイラは使用する計算機に依存しない部分（ソースプログラムを分析する機能をもつた部分）と，

どうしても計算機に依存せざるを得ない部分（分析結果をもとにしてオブジエクトプログラムを生成する機能をもつた部分等）に分離できる形に構成しておく。ここでは次の形をとつた。

コンパイラ
- 基本部分……機械無依存
- 機械記述部分……機械依存

機械無依存部分と機械依存部分を厳密に区分し，この方法の対象となる計算機の範囲をまつたく自由にしようとすれば，基本部分からの出力を処理する部分が大きく複雑になり，実用的でなくなる。そこで対象としてとりあげる機械の構造の相異の程度にある程度の制限をつければ，オブジエクトプログラムを生成する過程のうち共通的に扱かえ，かつその範囲内で機械無依存性を保てる部分が増える。この共通部分までを基本部分に含ませれば，機械記述部分は，

a）記号コードや語構成の記述（表）

b）機械語系の記述（プログラム）

となり，前述の(1)の作業はa）およびb）の書きかえ，またはb）のみの書きかえになり，(2)の作業はa）だけの置きかえに帰し，単純な作業となる。

2.3 実験結果

2.3.1 言　　語

置換法に使う言語としての条件は図2.1および図2.2からもわかるように，「機械無依存かつ自己記述可能（すなわちコンパイラがコンパイラ自身のソース言語で書けること）」ということである。このような条件に合致する言語が手近になかつたことなどのために，実験用のモデル言語を作つた。この言語はALGOLよりかなりレベルの低いもので，ALGOLのような手続き言語コンパイラのオブジエクト言語となることと，そのようなコンパイラを書く言語となることという2つの目的を考慮したもので，L_2 と呼んでいる。L_2 には入出力用の命令を含まない。これは機械依存度を大きくしないためと，入出力処理がオペレーティングシステムの機能としてまとめられることを考慮したためである。次にこの言語の概要を示す。

(1) L_2 のキヤラクタ

L_2 言語を機成するのは次のキヤラクタである。

アルファベット	A，B，…，Z
数　　字	0，1，…，9
記　　号	＋－／′＝．（）$＊，
ブ ラ ン ク	

(2) 命令形式

命令形式は図2.3に示す通りである。欄の意味は次の通りである。

LABEL　　プログラムポイントを表わす。第1文字が＊のときは行全体が注記行となる。

DELIMITER　操作の主要事項を宣言する。従来のアセンブラ言語のオペレーションコードに相当する。

SPECIFICATION　操作の明細を記入する。たとえば代数計算の場合，L_2 形式の数式が書かれる。

MELCOM L2 CODING FORM

PROBLEM			
LABEL	DELIMITER	SPECIFICATION	
1 2 6	8 13	15 20	
*			
* READ	ONE CHARACTER		
	DAT	O=CRA,	O=OCH, O=SIC,
	DAT	CAD=LIT	
RCH	INT	(LIT) −	(WDC) = (CRA) ,
	IFF	(R2H)	

図2.3　モデル言語L2の命令形式

(3)　オペランドの形式

情報単位はワードに限定する。ワードの長さや符号位置などは機械によつて異なるからL_2では定めない。

a）リテラル

１０進数	３．１２８など
８進数（論理語）	３\bar{O}１２（８進数１２）など
英数字列	$３ＡＸＹＺ（文字列ＸＹＺ）など
番地定数（番地語）	ＡＬＰＨＡなど，記号番地に与えられた絶対番地等価な数値である。

b）直接変数

絶対番地（１０進整数）または記号番地（アルファベットと数字の列）を一組のカッコでくくつて，その番地の内容をオペランドとすることを表わす。相対番地を表わす添字をつけてもよい。

例　（Ａ１２），（ＡＬＰＨＡ（（Ａ１２））

c）間接変数

直接変数をさらにもう一組のカッコでくくつて，間接番地的にオペランドをとり出すことを表わす。

例　（（Ａ１２）），（（ＢＥＴＡ（５）））

(4) 語　彙

コンパイラ動作の上からインストラクションとシュードオペレーションとに分割して考える。前者はオブジェクト言語に翻訳され実行されるもの，後者はコンパイラに対する指令である。L_2 言語の語彙とSPECIFICATION 部の例を表2.1に示す。INT および LOG で使えるオペレータは表2.2に示す通りである。

表2.1　モデル言語L2の語彙

	DELIMITER	機　能	SPECIFICATION の書き方	SPECIFICATION の例
インストラクション	INT	整数の四則および比較 (Integer)	$P_1\bar{O}P_2$ の形の操作指示要素をコンマでつないだもの．"P_1, P_2 の2つのオペランドに操作 $\bar{O}$ を実行せよ" を表わす．(表8は $\bar{O}$ の一覧表)．P_1 を欠くときは直前の結果が P_1 となる．$=R$ をつけ結果の宛先とすることもできる．	(A+B)→E, (A+B)×C/D と F との比較：(A)+(B)=(E), *(C), /(D), 'LT'(F)
	LOG	論理語演算 (Logical)		AVB→C：(A)+(B)=(C)
	IFF	比較の結果 "false" ならばとべ	飛先を表わす直接または間接変数	(JUMP) を INT の例の直後に置くと (A+B)×C/D≧F ならば JUMP へ飛ぶ
	GOT	無条件飛越		(JUMP)
	MGO	復帰点をセットして飛越	$ を先頭につけた番地定数	$SUB
	HAL	停止	直接または間接変数 (再開始点)	(RES)
シュードオペレーション	DAT	定数語をつくる，必要なら記号番地をつける	"リテラル" または "リテラル＝番地定数" の形をした要素をコンマでつないだもの	123=A, 124, 125 メモリに．123, 124, 125 のワードを作り先頭に記号番地 A をつける．
	REG	メモリを予約する	"番地定数 (整数定数)" の形をした要素をコンマでつないだもの	A 1(100), B(200) A 1として 100 ワード，B として 200 ワードを予約する．
	COM	コモンメモリを予約する		
	END	コンパイルの終了	計算の開始点を表わす直接変数	(START)

2.3.2　計　算　機

実験には計算機Xとして IBM−7090，計算機Yとして MELCOM−1530 を使った。前者は典型的な1アドレス計算機で，後者はストアドロジックを基本とする可変長命令形式をもつ計算機である。両計算機の違いのうちこの実験に関係する部分は次の通りである。

表2.2　INTおよびLOGで使えるオペレータ

DELIMITER	オ　ペ　レ　ー　タ
INT	+ (和) − (差) * (積) / (商) ** (2のベキ) // LT" GE" EQ' (<) (≧) (=)
LOG	+ (DR) * (AND) ** // (シフト)

a) プロセッサの比較，表2.3に示す。

b) キャラクタコードの比較

数字やアルファベットのコードは全く同じ。スペースおよび特殊記号の一部が異なる。

c) 命令形式の比較

図 2.4 に示す。

命令コードは全く異なっている。

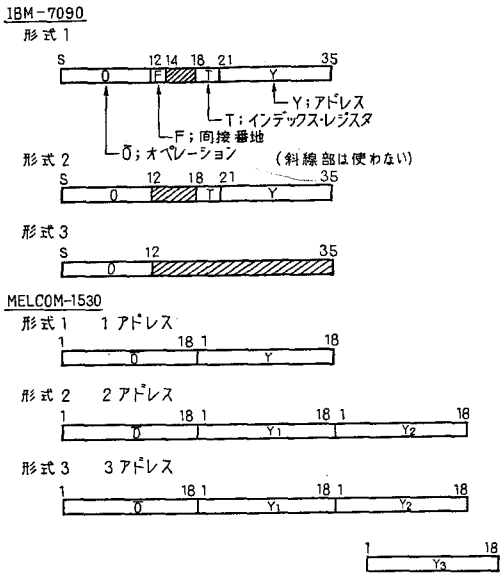


図 2.4 実験に用いた命令形式

表2.3 IBM－7090とMELCOM－1530のプロセッサの比較

項目		IBM 7090	MELCOM 1530
主メモリ			
	容量	32 K ワード	8 K ワード
	語長	36 ビット	18 ビット（または6ビット×3キャラクタ）
演算方式		2進	2進および2進化 10進
命令形式		単番地	ストアドロジック 可変長番地
インデックスレジスタ		3個	主メモリを使用
データ語の構成	2進語	1 符号 35ビット 絶対値	1 符号 17ビット 絶対値 (負数のときは補数形)
	論理語	36ビット	18ビット
	キャラクタ語	6ビット×6キャラクタ	6ビット×3キャラクタ

2.3.3 L_2 言語の解析

コンパイラを作成するに先立つて L_2 言語の解析を行なう。これは L_2 コンパイラの基本部分の翻訳プロセスを考えることである。

L_2 プログラムの最も大きな区切りはコーディングシートの1行である。各々の行はさらに3つの部分に分割されている。このうちのLABEL および DELIMITER 欄については簡単であり，SPECIFICATION 部についてのみ検討が必要である。

L_2 の語彙の中にはSPECIFICATION として複数個の要素を許すものがある。それらについては最も複雑な場合で

INTおよびLOGに対して　$P_1 \bar{O} P_2 = R$

DATに対して　リテラル＝番地定数

REGおよびCOMに対して　番地定数（整数定数）

の形の要素が現われるが，L_2 では要素相互間の関連を翻訳にあたつて考慮する必要のあるものはないから，1個の要素を処理の単位として差つかえない。各々の要素はコンマかブランクかによつて区切られている。

要素の意味はDELIMITERによつて確定されるから，DELIMITERごとに解析を行なえばよいが，インストラクションはすべてオブジエクト命令に変換されるので，一括して考える。

(1) インストラクション

インストラクションの各要素はオペレーション(通常DELIMITERできまる。ただしINTとLOGについてはDELIMITERとオペレータの2者によつてきまる。)とオペランドからなる。これらを次のような情報に分解する。

A. オペレーションに関する情報

要素が表わすオペレーションと等価なオブジエクト命令シーケンスを作り出すために必要な情報，あるいはそのような情報を索引するための情報。

B. オペランドに関する情報

a) 変数のタイプ

1重カツコ(直接変数)，2重カツコ(間接変数)の区別

b) Base アドレスのタイプ

10進数(絶対番地)，記号(一般メモリ…relocatable)，記号(コモンメモリ…common)，未定義の区別

c) Base アドレス

d) 添字のタイプ

内容はb)と同じ

e) 添字のアドレス

L_2 インストラクションの1要素にはオペランドを最大3個含むので以上の情報を3組用意し序列を明確にしておく必要がある。

(2) シュードオペレーション

シュードオペレーションはそれぞれ違つたコンパイル動作を要求しているのでDELIMITER ごとに違つた形の情報に分解しなければならない。DAT シュードオペレーションについて例をとると，その要素は次のように分解できる。

A. データのタイプ

整数データ，論理語データ，アルファニユメリックデータ，番地語データの区別。

B. データそのもの

C. その他の情報

a) 整数データの符号

b) 番地定数のタイプ

10進数，記号(一般メモリ… relocatable)，記号(コモンメモリ… common)，未定義の区別。

2.3.4 原形コンパイラの作成

置換え手順で用いる種となるコンパイラを次の手順で作成した。

(1) L_2 言語で7090 L_2 コンパイラを手書きする：

このコンパイラは，

A. 基本部分(Analyzing Program と呼ぶ)

B. 機械語系の記述(Composing Program と呼ぶ)

C. 記号のコード，語構成やオブジエクト命令の記述(Machine Descriptive Table と呼ぶ)

に分割して作成された。基本部分は前述の言語の解析に基いて作られている。表2.4～表2.5は，Analyzing Program から Composing Program に渡される情報の種類と内容をまとめたものである。この他に，共通に用いられるロケーションカウンタやコモンカウンタなどの変数類がある。表2.6はMachine Descriptive Tableの一覧である。

表2.4　基本部分から機械語命令生成部分に送られる情報(1)……インストラクションの場合

表の要素	情報の意味	内容
TMM 0	オペレーションコードとアドレス部の合成の要否	要=1，否=0
1	オペレーションコード抽出インデックス	
2	Base address の直接/間接変数の区別	直接=1，間接=0
3	Base address の Type	絶対番地=0 relocatable=1 common=2 未定義=−1
4	Base address	
5	添字の Type	絶対番地=0 relocatable=1 common=2 未定義=−1
6	添字	

表2.5　基本部分から機械語命令生成部分に送られる情報(2)……DATシュードオペレーションの場合

要素	情報の意味	内容
TYP	数値語のタイプ	整数=0 論理語=1 番地定数=2 キャラクタ=−1
SYL	数値(整数のときは絶対値だけ)	
SBT	整数のときの符号	負数=1,正数=0
ADT	番地定数の種類	絶対番地=0 relocatable=1 common=2 未定義=−1

表2.6　機械特性記述表

表の名称	内容
機械記述定数表	ワードおよびキャラクタの長さ，1ワードに格納するキャラクタ数，COMMON エリアの上限，1キャラクタ抽出のためのマスク，符号の位置，最低位の位置
外部コード表	L 2 キャラクタの外部コードの表（これと対応して内部コードの表が常設されている）
オペレーションコード表	オブジェクト命令シーケンスを形成するための情報

基本部分から機械語系の記述の部分へ送られる情報はソース言語を分析した結果だけであり，ソース言語の情報の形を表の形にかえたものにほかならない。

機械特性を与える情報のうち，基本部分で参照する必要があるのは表2.6に掲げたもので充分である。これらはソース言語の特性から作られ上述の Analyzing Program から Composing Program への情報の変換の過程で使用される。それ以外の機械特性情報はデータの形あるいはプログラム論理として機械語命令生成の部分で扱われる。

表2.6の内容が比較的簡単であるのは，前述のように対象となる

機械がある類形に含まれていることにもよる。しかしこれは7090と1530という仮定からこの形をとつたのではなく，Single Accumulator, Hommogeneous Addressingの機械という範囲ならばこの形となる。

(2) 上述のように作られたコンパイラをIBM−7090のアセンブラ言語であるFAP言語に書きかえる。この作業にはFAPのマクロアセンブル機能を応用した。

(3) (2)で作つたコンパイラをFAPアセンブラによつてIBM−7090の機械語に変換し原形コンパイラが完成した。

コンパイラの作成に要した労力は次の通りである。（実質延日数）

フローチャート作成	25日	合計46日
L_2 コーディング	7日	
FAP コーディング	9日	
デバギング	5日	

2.3.5 置　　換

置換実験は，図2.1および図2.2の両方について行なつたが，ここでは後者について記述する。

(1)

$$C^{L_{2,7090}\rightarrow L_{1530,1530}}_{L_2}$$ の作成

前述のコンパイラの機械語命令の生成部分と機械特性記述表のうちのオペレーションコードの部分をMELCOM−1530用におきかえてこれを得た。

(2) Step 1：原形コンパイラに上で得たコンパイラをソースプログラムとして与えて，IBM−7090で動作するMELCOM−1530用のオブジエクトプログラムを出すコンパイラ

$$C^{L_{2,7090}\rightarrow L_{1530,1530}}_{L_{7090,7090}}$$

を得た。

(3) Step 2：Step 1 によつて作られたコンパイラに，(1)で作成したコンパイラをソースプログラムとして再び与えて，

$$C^{L_{2,7090}\rightarrow L_{1530,1530}}_{L_{1530,1530}}$$

を得る。

(4) Step 3：Step 2 で作られたコンパイラは命令としてはMELCOM−1530の機械語が使われているが，コードやワード構成はIBM−7090用になつているため，(1)で置きかえなかつた部分をMELCOM−1530用にして，所要のコンパイラ

$$C^{L_{2,1530}\rightarrow L_{1530,1530}}_{L_{1530,1530}}$$

を得た。

表2.8は以上の実験過程で得た2つのコンパイラの諸元である。表中のIBM−7090機械語の部分は

$$C^{L_2\rightarrow L_{7090}}_{L_2} \quad \text{を} \quad C^{L_2\rightarrow L_{7090}}_{L_{7090}}$$

にかけてSelf-Compilationを行なつた結果についてのデータである。

この置換実験に要した労力は次の通りである。

$C^{L_{2,7090}\rightarrow L_{1530,1530}}_{L_2}$

の作成	4日
置換	2日
合計	6日

表2.8 L2コンパイラの諸元

	IBM-7090		MELCOM-1530	
	L2	Machine Language	L2	Machine Language
	ステップ	ワード	ステップ	ワード
外部コード表	7	54	7	54
オペレーションコード表	20	113	9	140
機械記述表	7	7	7	7
基本部分	517	1,386	534	2.632
機械語命令生成部分	21	55	40	341

またIBM－7090計算機使用時間は0.2時間である。置換で得たコンパイラも最初からアセンブラ言語で書いたとしたら原形コンパイラ作成時と同程度の労力を要したと思われるので，置換法によつて1/7以下の労力で別の機械のためのコンパイラが得られたことになる。

2.3.6 稼動実験

置換によつて作成したMELCOM－1530用のL_2コンパイラは入出力処理部分を含まないので，稼動にあたつてはこれを付加する必要がある。またMELCOM－1530はストアドロジック形の計算機であるので，L_2コンパイラのオブジェクトプログラムの基本機能を実現するマイクロプログラムが必要である。これらを整備した上で，マトリクス乗算プログラムおよび連立一次方程式の消去法による消法のプログラムについて稼動実験を行なつた。

プログラムは，比較のために，それぞれのALGOLプログラムを忠実にL_2およびSIA（MELCOM－1530のアセンブラ言語）で書き改めたために多分に冗長命令が含まれている。両者を対照させて表2.9に示した。この表から，置換によつて作られたコンパイラのオブジェクトプログラムは慣用のアセンブラ言語によるものと比較してメモリスペースおよび実行時間の両方で30％程余分に要したことがわかる。

表2.9　MELCOM－1530 L2 コンパイラの稼動実験

項目		連立方程式(20元)		マトリクス乗算(20×20)		(L2/SIA)×100
		L2	SIA	L2	SIA	
ソースプログラム		41 ステップ	90 ステップ	19 ステップ	42 ステップ	46%
オブジェクト・プログラムのスペース	データ	420 ワード	420 ワード	1200 ワード	1200 ワード	100
	プログラム(作業番地含む)	319 ワード	254 ワード	111 ワード	95 ワード	126
所要時間	コンパイル	25 秒	30 秒	14 秒	15 秒	87
	実行	28 秒	22 秒	14 秒	11 秒	127

注　連立方程式プログラムは特定の係数および定数を準備し消去法によって解いた時間の合計である．

2 4　結　　言

計算機を用いてコンパイラを作り出し，それによつてコンパイルしたプログラムを実行するという過程について，置換を中心とする方法を案出し，実験した結果を示した。

最後にL_2 コンパイラのリステイングを添付しておく。

$$C\begin{matrix}L_{2,7090}\rightarrow L_{1530,1530}\\ L_2\end{matrix}$$

については全体を示し，その中の一部分について対応する部分を

$$C\begin{matrix}L_{2,7090}\rightarrow L_{1530,1530}\\ L_{7090}\end{matrix}$$

および

$$C\begin{matrix}L_{2,7090}\rightarrow L_{1530,1530}\\ L_{1530}\end{matrix}$$

の両者からとり出して示す。

第3章　結　　論

計算機ソフトウエア開発のために計算機を活用することの一つとして，計算機X用のコンパイラを計算機Xを用いて計算機Y用のコンパイラに変える問題をとりあげ，その解決法として置換法による一連の手続きを案出した。L_2 と呼ばれる中間言語をモデルに選んでこれに対して置換法を適用し，実際にIBM−7090およびMELCOM−1530を使用してコンパイラの自動生成を行ない，さらに得られたコンパイラで二，三の例題を処理させてオブジエクトプログラムを実行させた。

この過程を通じて機械処理にかけることを前提としたコンパイラの作成技法を追求し，また，異機種間のコンパイラ書きかえに言語の相異と並んで生じるコードの相異の問題についても検討を加えた。

この実験ではコンパイラの作成所要時間を充分短縮することができたが，これは主としてコンパイラを機械に依存する部分とそうでない部分とに容易に分割し得る形で構成しておいたことに基くと思われる。

この方法では実用的な制限を考えてあるので，対象とする機械の構造の相異が大きくなると機械無依存の性質を保つて基本部分に組み入れ得る部分が少なくなり，置きかえ過程が複雑になるという問題点がある。また，実験で用いたモデル言語 L_2 の構造による制約として，2進計算機であること，種となるコンパイラに使われている情報単位を1語に収容できること，機械Xは機械Yより語長が大きいことが条件となつている。

このような方法で得られたコンパイラの生み出すオブジエクトプログラムの効率については，実験では実行時間1.3倍という値が得られているが，この増加の原因は主として実験に用いたオブジエクト言語の語彙が多くなく，きめのこまかい処理をしていないことにあると考えられる。効率向上についてはまだかなりの可能性がある。

(1)　L_2　コンパイラ・リスティング

```
C L2,7090 ⟶ L1530,1530
  L2
```

```
INPUT DATA          * M428F2041 112 M04113 005

*********0*********0*********0*********0*********0*********0*********0*********0

L2BCCC                                                                    L2B   11
  653                                                                     L2B   21
*               L2 COMPILER  MODEL (MACHINE INDEPENDENT)                  L2B    3
*                                                                         L2B    4
*                                                                         L2B    5
*               EXTERNAL CODE TABLE (47 WORDS)                            L2B    6
        DAT     60=TEX,28,59,11,16,32,49,44,43,27,12          (),=+-/*$.' L2B   71
        DAT     0,1,2,3,4,5,6,7,8,9                           NUMBER(0-9) L2B    8
        DAT     38=OU,17,18,19,20,21,22,23,24,25,33           ALPHA (OA-J)L2B    9
        DAT     34,35,36,37,39,40,41,50,51                    ALPHA (K-T) L2B   10
        DAT     52,53,54,55,56,57                             ALPHA (U-Z) L2B   11
        DAT     48=SPS                                        SPACE       L2B   121
        DAT     60=QOP,11=QEQ,32=QMI,49=QSL,44=QAS,43=QDL                 L2B   131
*                                                                         L2B   14
*               MACHINE DEPENDENT TABLE                                   L2B   141
* ENTRY TABLE FOR MACHINE OPERATION CODE TABLE                            L2B 598
        DAT     1=TCM,4,7,9,12,14,17,25,33,19,27,35,41,43,46,48,55,57,60  L2B 599
        DAT     62,1,4,7,9,0,0,0,0,0,0,0,0,65,67,0,0,74,76,79,81          L2B 600
        DAT     84=GNC,87=GMN,92=HNC                                      L2B 601
* MACHINE OPERATION CODE TABLE                                           L2B 602
        DAT     0=TOP,1,7173,7,1,7176,7,1,7176,1,7186,7,1,7176,1,7194,7,1 L2B 603
        DAT     7176,1,7194,7,1,7288,7,1,7176,1,7194,7,1,7295,7,1,7176,1  L2B 604
        DAT     7194,7,1,7280,7,1,7176,1,7202,7,1,7176,0,7183,0,7287,1    L2B 605
        DAT     7233,7,1,7176,1,7302,7,1,7176,1,7332,7,1,7176,1,7179,0    L2B 606
        DAT     43,0,43,7,1,7176,1,7352,7,1,7176,1,7365,7,1,7150,7,1,7153 L2B 607
        DAT     0,43,7,0,7157,0,0,7                                       L2B 608
*                                                                         L2B   149
*               MACHINE DESCRIPTIVE CONSTANTS                             L2B   15
        DAT     3=CPW                 CHARACTERS PER ONE WORD             L2B   16
        DAT     6=CHL                 BIT LENGTH OF ONE CHARACTER         L2B   17
        DAT     0=UPW                 UNIT POSITION ON WORD               L2B   18
        DAT     $60400000=SIB         SIGN BIT                            L2B   19
        DAT     $101=LSB              LEAST SIGNIFICANT BIT               L2B   20
        DAT     7149=UL               HIGHEST LOCATION FOR COMMON AREA    L2B   21
        DAT     $2077=EC1             MASK TO EXTRACT ONE CHARACTER       L2B   22
* TABLE                                                                   L2B   23
*               BINARY OPERATON TABLE                                     L2B   24
        DAT     $30123=TBO,$30001,$30307,$30107,$30311,$30111,$30305      L2B   25
        DAT     $30301,$30303,$30105,$30101,$30103,$30315,$30115,$30313   L2B   26
        DAT     $30113,$30317,$30117,$30321,$30121                        L2B   27
        DAT     20=LBO                        NUMBER OF TBP               L2B   28
*               RELATIONAL OPERATION TABLE                                L2B   29
        DAT     $3AEQ =TRL,$3ALT ,$3AGE ,0                                L2B   30
        DAT     34=EQY,32=LTY,33=GEY                                      L2B   31
        DAT     40=YDS,39=YDA                 YDS=//    YAS=**          L2B   32
*               DELIMITER TABLE                                           L2B   33
        DAT     $3AINT=TDL,$3ALCG,$3AGOT,$3ADAT,$3AIFF,$3AMGO,$3AREG      L2B   34
        DAT     $3ACOM,$3AEND,$3AHLT,0                                    L2B   35
*                                                                         L2B   351
*               L2 COMPILER BASIC PROGRAM                                 L2B   36
* CONSTANTS                                                              L2B   37
        DAT     0=ZER,1=ONE,2=TWO,3=THR,4=FOR,5=FIV,6=SIX,7=SEV,10=TEN    L2B   38
        DAT     14=C14,15=C15,72=C72,7=EDM                                L2B   39
* VARIABLES AND AREAS                                                    L2B   40
        DAT     0=BLK,0=BKC,0=CSC                                         L2B   41
        DAT     0=WDC,0=CCC,0=CDC,0=SMC,0=LCC,0=MSC,0=CMC,0=ERC,0=CHC     L2B   42
        DAT     0=TMM,0=QM1,0=QM2,0=QM3,0=QM4,0=QM5,0=QM6                 L2B   43
```

```
*********0*********0*********0*********0*********0*********0*********0*********0
        REG     TCO(50),TSM(1000),TMS(500),TER(200),TAD(15),TOM(20)        L2B  44
        COM     NCR(1),CBC(1),CAD(24),OB1(1),OB2(1),OB3(1),DUP(1)          L2B  45
* INITIAL CONDITION SETTING                                                L2B  46
BEG     INT     (ZER)=(CCC),=(SMC),=(LCC),=(CDC),=(ERC),=(NCR),=(DUP)      L2B  47
        INT     =(CBC),=(MSC),=(CSC),=(BLK),(UL)=(CMC),(CPW)=(BKC)         L2B  48
BE1     LOG     (BLK)**(CHL),+(SPS)=(BLK)          PRODUCE BLANK WORD      L2B  49
        INT     (BKC)-(ONE)=(BKC),'EQ'(ZER)                                L2B  50
        IFF     (BE1)                                                      L2B  51
        GOT     (SRC)                                                      L2B  52
* SUBROUTINES                                                             L2B  53
* STACKING ERROR INFORMATIONS                                             L2B  54
        DAT     0=ER,0=MER                                                 L2B  551
ERR     INT     (ER)=(TER((ERC))),(ERC)+(ONE)=(ERC),(CDC)=(TER((ERC)))     L2B  56
        INT     (ONE)+(ERC)=(ERC)                                          L2B  57
        GOT     ((MER))                                                    L2B  581
* READ ONE CHARACTER                                                      L2B  59
        DAT     0=SIC,0=CRA,0=OCH,0=MRC                                    L2B  60
        DAT     CAD=LIT                                                    L2B  61
RCH     INT     (LIT)-(WDC)=(CRA),(CHC)+(ONE),'LT'(CPW)                    L2B  62
        IFF     (R2H)                                                      L2B  63
        INT     (CPW)-(ONE),-(CHC),*(CHL)=(SIC)                            L2B  64
        LOG     ((CRA))//(SIC),*(EC1)=(OCH)                                L2B  65
        INT     (ONE)+(CHC)=(CHC)                                          L2B  66
R3H     INT     (ONE)+(CCC)=(CCC)                                          L2B  67
        GOT     ((MRC))                                                    L2B  68
R2H     LOG     ((CRA))*(EC1)=(OCH)                                        L2B  69
        INT     (ONE)+(WDC)=(WDC),(ZER)=(CHC)                              L2B  70
        GOT     (R3H)                                                      L2B  71
* GIVE ATTRIBUTE                                                          L2B  72
* INTERNAL PATTERN TABLE (47 WORDS)                                       L2B  73
        DAT     42=YOP,46,43                                 (),=+-/*$.'   L2B  74
        DAT     41=YEQ,35=YPL,36=YMI,37=YSL,38=YAS,44=YDL,45,47            L2B  75
        DAT     1,1,1,1,1,1,1,1,1,1                                NUMBER  L2B  76
        DAT     2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,0          L2B  77
        DAT     0=GV1,0=ATT,0=MGA                                          L2B  78
GVA     INT     (ZER)=(GV1)                                                L2B  79
GV3     INT     (TEX((GV1)))'EQ'(OCH)                                      L2B  80
        IFF     (GV2)                                                      L2B  81
        INT     (YOP((GV1)))=(ATT)                                         L2B  82
        GOT     ((MGA))                                                    L2B  83
GV2     INT     (ONE)+(GV1)=(GV1)                                          L2B  84
        GOT     (GV3)                                                      L2B  85
* READ SYLLABLE                                                           L2B  86
        DAT     0=SYL,0=SLC,0=EM,0=SPC,0=LET,0=NCH,0=NAT,0=LAT,0=MSY       L2B  87
        DAT     RR1=MR1,RR2=MR2                                            L2B  88
RSL     INT     (ZER)=(SYL),=(EM),=(SLC),=(SPC),=(ATT),(ONE)=(LET)         L2B  89
        INT     (NCH)'EQ'(ZER)                                             L2B  90
        IFF     (RS2)                                                      L2B  91
R2S     INT     (CCC)'LT'(C72)                                             L2B  92
        IFF     (R3S)                                                      L2B  93
        INT     (MR1)=(MRC)                                                L2B  94
        GOT     (RCH)                                                      L2B  95
RR1     INT     (ATT)=(LAT),(MR2)=(MGA)                                    L2B  96
        GOT     (GVA)                                                      L2B  97
RR2     INT     (ATT)'EQ'(ZER)                                             L2B  98
        IFF     (R7S)                                                      L2B  99
        INT     (SLC)'EQ'(ZER)                                             L2B 100
        IFF     (R3S)                                                      L2B 101
        INT     (SPC)'EQ'(FIV)                                             L2B 102
```

```
*********0*********0*********0******+***0*********+0*********0*********0+********0
         IFF    (R6S)                                                          L2B 103
 R3S     INT    (ONE)=(EM)                                                     L2B 104
 R4S     INT    (LET)'EQ'(ONE)                                                 L2B 105
         IFF    (RES)                                                          L2B 106
 R5S     INT    (SLC)'LT'(CPW)                                                 L2B 107
         IFF    (RES)                                                          L2B 108
         LCG    (SYL)**(CHL),+(SPS)=(SYL)                                      L2B 109
         INT    (ONE)+(SLC)=(SLC)                                              L2B 110
         GOT    (R5S)                                                          L2B 111
 R6S     INT    (ONE)+(SPC)=(SPC)                                              L2B 112
         GOT    (R2S)                                                          L2B 113
 R7S     INT    (ATT)'GE'(THR)                                                 L2B 114
         IFF    (CNS)                                                          L2B 115
         INT    (SLC)'EQ'(ZER)                                                 L2B 116
         IFF    (R9S)                                                          L2B 117
 R8S     INT    (ATT)=(SYL)                                                    L2B 118
 RES     GOT    ((MSY))                                                        L2B 119
 R9S     INT    (OCH)=(NCH),(ATT)=(NAT),(LAT)=(ATT)                            L2B 120
         GOT    (R4S)                                                          L2B 121
 RS2     INT    (NCH)=(OCH),(ZER)=(NCH),(NAT)=(ATT)                            L2B 122
         GOT    (R8S)                                                          L2B 123
 CNS     INT    (SLC)'EQ'(ZER)                                                 L2B 124
         IFF    (C2S)                                                          L2B 125
         INT    (ATT)'EQ'(ONE)                                                 L2B 126
         IFF    (C2S)                                                          L2B 127
         INT    (ZER)=(LET)                                                    L2B 128
 C2S     INT    (LET)'EQ'(ZER)                                                 L2B 129
         IFF    (C4S)                                                          L2B 130
         INT    (SYL)*(TEN),+(OCH)=(SYL)                                       L2B 131
 C3S     INT    (CNE)+(SLC)=(SLC)                                              L2B 132
         GOT    (R2S)                                                          L2B 133
 C4S     INT    (SLC)'LT'(CPW)                                                 L2B 134
         IFF    (R2S)                                                          L2B 135
         LOG    (SYL)**(CHL),+(CCH)=(SYL)                                      L2B 136
         GOT    (C3S)                                                          L2B 137
 * REGISTER SYMBOL                                                            L2B 138
         DAT    0=IDF,0=MRM                                                    L2B 139
 RSM     INT    (IDF)=(TSM((SMC))),(CNE)+(SMC)=(SMC)                           L2B 140
         INT    (LCC)=(TSM((SMC))),(ONE)+(SMC)=(SMC)                           L2B 141
         GOT    ((MRM))                                                        L2B 142
 * SET WORD COUNTER AND CHARACTER COUNTER                                     L2B 143
         DAT    0=MSW                                                          L2B 144
 WCS     INT    (C15)=(CCC)                                                    L2B 145
         GOT    (WC1)                                                          L2B 146
 SWC     INT    (C14)=(CCC)                                                    L2B 147
 WC1     INT    (CCC)/(CPW)=(WDC),*(CPW)=(CHC),(CCC)-(CHC)=(CHC)               L2B 148
         GOT    ((MSW))                                                        L2B 149
 * END OF SUBROUTINES                                                         L2B 150
 *                                                                           L2B 151
 * READ ONE CARD                                                             L2B 152
 SRC     MGO    SROC                    ROC=INPUT SUBROUTINE                   L2B 1531
         GOT    (LBL)                   RETURN POINT FROM INPUT SUBROUTINE     L2B 154
 * PROCESSING LABEL PART                                                      L2B 155
         DAT    RLB=MLB,DEL=L4L                                                L2B 156
 LBL     INT    (ZER)=(WDC),=(CHC),=(NCH),=(CCC),(ONE)+(CDC)=(CDC)             L2B 157
         INT    (MLB)=(MSY)                                                    L2B 158
         GOT    (RSL)                                                          L2B 159
 RLB     INT    (ATT)'LT'(THR)                                                 L2B 160
         IFF    (SRC)                                                          L2B 161
```

```
*********0*********0*********0*********0********+0*********0*********0*+*******0
         INT    (SYL)'EQ'(BLK)                                              L2B 162
         IFF    (L2L)                                                       L2B 163
L3L      GOT    (DEL)                                                       L2B 164
L2L      INT    (SYL)=(IDF),(L4L)=(MRM)                                     L2B 165
         GOT    (RSM)                                                       L2B 166
* PROCESSING DELIMITER PART                                                 L2B 167
         DAT    0=DLC                                                       L2B 168
         DAT    R2L=M2L,SRC=M3L,D5L=D6L                                     L2B 169
         DAT    PIT=TET,PLG,PGT,PDA,PIF,PGM,PRG,PCM,PED,PHT                 L2B 170
DEL      INT    (SEV)=(CCC),(D6L)=(MSW)                                     L2B 171
         GOT    (WC1)                                                       L2B 172
D5L      INT    (M2L)=(MSY)                                                 L2B 173
         GOT    (RSL)                                                       L2B 174
R2L      INT    (ZER)=(DLC)                                                 L2B 175
D2L      INT    (TDL((DLC)))'EQ'(SYL)                                       L2B 176
         IFF    (D3L)                                                       L2B 177
         GOT    ((TET((DLC))))                                              L2B 178
D3L      INT    (TDL((DLC)))'EQ'(ZER)                                       L2B 179
         IFF    (D4L)                                                       L2B 180
         INT    (SYL)=(ER)                                                  L2B 181
         INT    (M3L)=(MER)                                                 L2B 182
         GOT    (ERR)                                                       L2B 183
D4L      INT    (ONE)+(DLC)=(DLC)                                           L2B 184
         GOT    (D2L)                                                       L2B 185
* RESERVING MEMORY BLOCK                                                    L2B 186
         DAT    R2G=M2G,R3G=M3G,M9G=M4G,R5G=M5G,R6G=M6G                     L2B 187
         DAT    0=REG,0=MMB                                                 L2B 188
RMB      INT    (M2G)=(MSW)                                                 L2B 189
         GOT    (SWC)                                                       L2B 190
R2G      INT    (M3G)=(MSY)                                                 L2B 191
         GOT    (RSL)                                                       L2B 192
R3G      INT    (SYL)=(IDF)                                                 L2B 193
R4G      INT    (ZER)=(NCH),(M5G)=(MSY)                                     L2B 194
         GOT    (RSL)                                                       L2B 195
R5G      INT    (REG)'EQ'(ONE)                                              L2B 196
         IFF    (R7G)                                                       L2B 197
         INT    (M4G)=(MRM)                                                 L2B 198
         GOT    (RSM)                                                       L2B 199
M9G      INT    (SYL)+(LCC)=(LCC)                                           L2B 200
R8G      INT    (ZER)=(NCH),(M6G)=(MSY)                                     L2B 201
         GOT    (RSL)                                                       L2B 202
R6G      INT    (EM)'EQ'(ONE)                                               L2B 203
         IFF    (R2G)                                                       L2B 204
         INT    (REG)'EQ'(ONE)                                              L2B 205
         IFF    (R9G)                                                       L2B 206
         INT    (ONE)=(DUP)                                                 L2B 207
         MGO    SMOU                                                        L2B 2081
         INT    (ZER)=(DUP)                                                 L2B 209
R9G      GOT    ((MMB))                                                     L2B 210
R7G      INT    (IDF)=(TCO((CSC))),(ONE)+(CSC)=(CSC)                        L2B 211
         INT    (LCC)=(TCO((CSC))),(ONE)+(CSC)=(CSC),(LCC)-(SYL)=(LCC)      L2B 212
         GOT    (R8G)                                                       L2B 213
* PROCESSING REGION                                                         L2B 214
         DAT    SRC=MPG                                                     L2B 215
PRG      INT    (ONE)=(REG),(MPG)=(MMB)                                     L2B 216
         GOT    (RMB)                                                       L2B 217
* PROCESSING COMMON                                                         L2B 218
         DAT    0=LCT,RPC=MPC                                               L2B 219
PCM      INT    (ZER)=(REG),(LCC)=(LCT),(CMC)=(LCC),(MPC)=(MMB)             L2B 220
```

```
*********0*********0*********0*********0*********0*********0*********0*********0
         GOT    (RMB)                                                        L2B 221
RPC      INT    (LCC)=(CMC),(LCT)=(LCC)                                      L2B 222
         GOT    (SRC)                                                        L2B 223
* OUTPUT ONE ML                                                             L2B 224
         DAT    0=OP,0=TB,0=MSO                                              L2B 225
SOC      INT    (LCC)=(OB1),+(ONE)=(LCC),(OP)=(OB2),(TB)=(OB3)               L2B 226
         MGO    $MOU                                                         L2B 227
         GOT    ((MSC))                                                      L2B 228
* PROCESSING $NT CONSTANT                                                   L2B 229
         DAT    0=DC,0=IDC,0=MDT                                             L2B 230
         DAT    R2N=M2N,R3N=M3N,R4N=M4N,D7T=M5N                               L2B 231
DNT      INT    (ZER)=(DC)                                                   L2B 232
D2T      INT    (M2N)=(MRC)                                                  L2B 233
         GOT    (RCH)                                                        L2B 234
R2N      INT    (M3N)=(MGA)                                                  L2B 235
         GOT    (GVA)                                                        L2B 236
R3N      INT    (ATT)'EQ'(ONE)                                               L2B 237
         IFF    (D3T)                                                        L2B 238
         INT    (DC)*(TEN),+(OCH)=(DC)                                       L2B 239
         GOT    (D2T)                                                        L2B 240
D3T      INT    (ZER)=(SYL),(OCH)'EQ'(OU)                                    L2B 241
         IFF    (D6T)                                                        L2B 242
D4T      INT    (DC)'EQ'(ZER)                OCTAL DATA                      L2B 243
         IFF    (D5T)                                                        L2B 244
DET      GOT    ((MDT))                                                      L2B 245
D5T      INT    (M4N)=(MRC)                                                  L2B 246
         GOT    (RCH)                                                        L2B 247
R4N      LOG    (SYL)**(THR),+(OCH)=(SYL)                                    L2B 248
         INT    (DC)-(ONE)=(DC)                                              L2B 249
         GOT    (D4T)                                                        L2B 250
D6T      INT    (ZER)=(IDC)                                                  L2B 251
D9T      INT    (ZER)'LT'(DC)                                                L2B 252
         IFF    (D8T)                                                        L2B 253
         INT    (M5N)=(MRC)                                                  L2B 254
         GOT    (RCH)                                                        L2B 255
D7T      INT    (DC)-(ONE)=(DC),(IDC)'LT'(CPW)                               L2B 256
         IFF    (D9T)                                                        L2B 257
D1T      LOG    (SYL)**(CHL),+(OCH)=(SYL)                                    L2B 258
         INT    (CNE)+(IDC)=(IDC)                                            L2B 259
         GOT    (D9T)                                                        L2B 260
D8T      INT    (IDC)'LT'(CPW)                                               L2B 261
         IFF    (DET)                                                        L2B 262
         INT    (SPS)=(CCH)                                                  L2B 263
         GOT    (D1T)                                                        L2B 264
* ASSIGNING ADDRESS                                                         L2B 265
         DAT    0=TDC,0=KID,0=ABA,0=MAS,0=SNC,0=SFG                          L2B 266
ASS      INT    (ZER)=(SNC)                                                  L2B 267
A2S      INT    (TSM((SNC)))'EQ'(KID)                                        L2B 268
         IFF    (A4S)                                                        L2B 269
         INT    (ONE)+(SNC)=(SNC),(TSM((SNC)))=(ABA),(ONE)=(SFG)             L2B 270
A3S      GOT    ((MAS))                                                      L2B 271
A4S      INT    (TWO)+(SNC)=(SNC),'GE'(SMC)                                  L2B 272
         IFF    (A2S)                                                        L2B 273
         INT    (ZER)=(SNC)                                                  L2B 274
A6S      INT    (TCO((SNC)))'EQ'(KID)                                        L2B 275
         IFF    (A5S)                                                        L2B 276
         INT    (ONE)+(SNC)=(SNC),(TCO((SNC)))=(ABA),(TWO)=(SFG)             L2B 277
         GOT    (A3S)                                                        L2B 278
A5S      INT    (TWO)+(SNC)=(SNC),'GE'(CSC)                                  L2B 279
```

```
*********0++*******0********0*******+**0********+0********0********0********0
          IFF     (A6S)                                                          L2B 280
          INT     (ZER)-(CNE)=(SFG),(KID)=(ABA)                                  L2B 281
          GOT     (A3S)                                                          L2B 282
    * PROCESSING DATA                                                           L2B 283
          DAT     R2D=M2D,R3D=M3D,R4D=M4D,R5D=M5D,R6D=M6D,R7D=M7D,R8D=M8D        L2B 284
          DAT     RD3=MD3,R9D=M9D                                                L2B 285
          DAT     0=SBT,0=MGC                                                    L2B 286
    PDA   INT     (M2D)=(MSW)                                                    L2B 287
          GOT     (SWC)                                                          L2B 288
    R2D   INT     (ZER)=(NCH),(M3D)=(MSY)                                        L2B 289
          GOT     (RSL)                                                          L2B 290
    R3D   INT     (ATT)'EQ'(YDL)                                                 L2B 291
          IFF     (D2A)                                                          L2B 292
          INT     (M4D)=(MDT)                                                    L2B 293
          GOT     (DNT)                                                          L2B 294
    R4D   INT     (SYL)=(OP),(ZER)=(TB)                                          L2B 295
    MD4   INT     (M6D)=(MSY)                                                    L2B 296
          GOT     (RSL)                                                          L2B 297
    R6D   INT     (ATT)'EQ'(YEQ)                                                 L2B 298
          IFF     (R8D)                                                          L2B 299
          INT     (M7D)=(MSY)                                                    L2B 300
          GOT     (RSL)                                                          L2B 301
    R7D   INT     (SYL)=(IDF),(M8D)=(MRM)                                        L2B 302
          GOT     (RSM)                                                          L2B 303
    R8D   INT     (M5D)=(MSO)                                                    L2B 304
          GOT     (SOC)                                                          L2B 305
    R5D   INT     (EM)'EQ'(ONE)                    EM=END MARK                   L2B 306
          IFF     (R2D)                                                          L2B 307
          GOT     (SRC)                                                          L2B 308
    D2A   INT     (ATT)'EQ'(YMI)                                                 L2B 309
          IFF     (MD2)                                                          L2B 310
          INT     (M9D)=(MSY)                                                    L2B 311
          GOT     (RSL)                                                          L2B 312
    R9D   INT     (SIB)=(SBT)                                                    L2B 313
    MD8   INT     (M4D)=(MGC)                                                    L2B 314
          GOT     (GCW)                            TO AUXILIARY PROGRAM          L2B 315
    MD7   INT     (ZER)=(SBT)                                                    L2B 316
          GOT     (MD8)                                                          L2B 317
    MD2   INT     (LET)'EQ'(ONE)                                                 L2B 318
          IFF     (MD7)                                                          L2B 319
          INT     (SYL)=(KID),(MD3)=(MAS)                                        L2B 320
          GOT     (ASS)                                                          L2B 321
    RD3   INT     (SFG)'LT'(ZER)                                                 L2B 322
          IFF     (MD5)                                                          L2B 323
          INT     (ZER)=(OP),=(TB),(ABA)=(TMS((MSC)))                            L2B 324
          INT     (CNE)+(MSC)=(MSC),(LCC)=(TMS((MSC))),(ONE)+(MSC)=(MSC)         L2B 325
          GOT     (MD4)                                                          L2B 326
    MD5   INT     (SFG)=(TB),(ABA)=(OP)                                          L2B 327
          GOT     (MD4)                                                          L2B 328
    * PROCESSING CNE OPERAND                                                    L2B 329
          DAT     0=PC,0=MOA                                                     L2B 330
          DAT     R20=M20,R30=M30,R40=M40,R50=M50,R60=M60,R10=M70,03A=M80         L2B 331
          DAT     R90=M90                                                        L2B 332
    POA   INT     (ZER)=(PC),(M20)=(MSY)                                         L2B 333
          GOT     (RSL)                                                          L2B 334
    R20   INT     (ATT)'EQ'(YOP)                                                 L2B 335
          IFF     (09A)                                                          L2B 336
          INT     (ONE)=(TAD((TDC))),+(PC)=(PC),(M30)=(MSY)                      L2B 337
          GOT     (RSL)                                                          L2B 338
```

```
*********0+********0*********0*******+**0********+*0*********0*********0*+********0
R30     INT     (ONE)+(TDC)=(TDC),(LET)'EQ'(ZER)                                L2B 339
        IFF     (08A)                                                           L2B 340
        INT     (SYL)=(ABA)                                                     L2B 341
02A     INT     (LET)=(TAD((TDC))),(ONE)+(TDC)=(TDC),(ABA)=(TAD((TDC)))         L2B 342
        INT     (CNE)+(TDC)=(TDC),(M4C)=(MSY)                                   L2B 343
        GOT     (RSL)                                                           L2B 344
R40     INT     (ATT)'EQ'(YOP)                                                  L2B 345
        IFF     (07A)                                                           L2B 346
        INT     (CNE)+(PC)=(PC),(M5C)=(MSY)                                     L2B 347
        GOT     (RSL)                                                           L2B 348
R50     INT     (ATT)'EQ'(YOP)                                                  L2B 349
        IFF     (06A)                                                           L2B 350
        INT     (CNE)+(PC)=(PC),(M6C)=(MSY)                                     L2B 351
        GOT     (RSL)                                                           L2B 352
R60     INT     (SYL)=(KID),(ONE)=(TAD((TDC))),(M7O)=(MAS)                      L2B 353
        GOT     (ASS)                                                           L2B 354
R10     INT     (SFG)=(TAD((TDC)))                                              L2B 355
R70     INT     (ONE)+(TDC)=(TDC),(ABA)=(TAD((TDC))),(ONE)+(TDC)=(TDC)          L2B 356
03A     INT     (PC)'EQ'(ZER)                                                   L2B 357
        IFF     (05A)                                                           L2B 358
        INT     (ZER)=(NCH)                                                     L2B 359
        GOT     ((MOA))                                                         L2B 360
05A     INT     (PC)-(ONE)=(PC),(M8O)=(MRC)                                     L2B 361
        GOT     (RCH)                                                           L2B 362
06A     INT     (SYL)=(ABA),(ZER)=(TAD((TDC)))                                  L2B 363
        GOT     (R7O)                                                           L2B 364
07A     INT     (ZER)=(TAD((TDC))),(ONE)+(TDC)=(TDC),(ZER)=(TAD((TDC)))         L2B 365
        INT     (ONE)+(TDC)=(TDC)                                               L2B 366
        GOT     (03A)                                                           L2B 367
08A     INT     (SYL)=(KID),(M9C)=(MAS)                                         L2B 368
        GOT     (ASS)                                                           L2B 369
R90     INT     (SFG)=(LET)                                                     L2B 370
        GOT     (02A)                                                           L2B 371
09A     INT     (ZER)=(TAD((TDC)))                                              L2B 372
        GOT     (R30)                                                           L2B 373
* PROCESSING UNDEFINED BASE ADDRESS                                            L2B 374
        DAT     0=GT,0=MUD,0=LC,0=UND,U3D=U2D                                   L2B 375
PUD     INT     (QM3)'LT'(ZER)                                                  L2B 376
        IFF     (U4D)                                                           L2B 377
        INT     (GT)'EQ'(CNE)                                                   L2B 378
        IFF     (U5D)                                                           L2B 379
        INT     (QM4)=(TMS((MSC))),(ONE)+(MSC)=(MSC)                            L2B 380
        INT     (LC)=(TMS((MSC))),(ONE)=(UND)                                   L2B 381
U30     INT     (ZER)=(QM3),=(QM4)                                              L2B 382
U4D     GOT     ((MUD))                                                         L2B 383
U5D     INT     (QM4)=(ER),(U2D)=(MER)                                          L2B 384
        GOT     (ERR)                                                           L2B 385
* PROCESSING MACHINE INSTRUCTION                                               L2B 386
        DAT     0=MML,0=OMC,0=MIX,0=MID,0=MOP,0=OLC                             L2B 387
        DAT     R2I=M2I,R3I=M3I,R4I=M4I,R5I=M5I,R6I=M6I,R7I=M7I,MI6=M8I         L2B 388
PML     INT     (TMM)'EQ'(ONE)                                                  L2B 389
        IFF     (MI5)                                                           L2B 390
        INT     (ZER)=(OLC),=(UND),(LCC)=(LC),(QM5)'LT'(ZER)                     L2B 391
        IFF     (MI4)                                                           L2B 392
        INT     (QM6)=(ER),(M2I)=(MER)                                          L2B 393
        GOT     (ERR)                                                           L2B 394
R2I     INT     (ZER)=(QM6),=(QM5)                                              L2B 395
MI2     INT     (M3I)=(MIX)                                                     L2B 396
        GOT     (PIX)                          TO AUXILIARY PROGRAM             L2B 397
```

```
*********0*********0*********0*********0*********0*********0*********0*********0
R3I     INT     (QM2)'EQ'(ONE)                                                  L2B 398
        IFF     (R4I)                                                           L2B 399
        INT     (M4I)=(MID)                                                     L2B 400
        GOT     (PID)                           TO AUXILIARY PROGRAM            L2B 401
R4I     INT     (M5I)=(MOP)                                                     L2B 402
        GOT     (POP)                           TO AUXILIARY PROGRAM            L2B 403
R5I     INT     (UND)'EQ'(ONE)                                                  L2B 404
        IFF     (RI)                                                            L2B 405
        INT     (ONE)+(MSC)=(MSC)                                               L2B 406
RI      INT     (ZER)=(OMC)                                                     L2B 407
MI3     INT     (TOM((OMC)))=(TE),(ONE)+(OMC)=(OMC),(TOM((OMC)))=(CP)           L2B 408
        INT     (M6I)=(MSO)                                                     L2B 409
        GOT     (SOC)                                                           L2B 410
R6I     INT     (ONE)+(OMC)=(OMC),'GE'(OLC)                                     L2B 411
        IFF     (MI3)                                                           L2B 412
MI6     GOT     ((MML))                                                         L2B 413
MI4     INT     (QM5)'EQ'(ZER)                                                  L2B 414
        IFF     (MI2)                                                           L2B 415
        INT     (M7I)=(MUD)                                                     L2B 416
        GOT     (PUD)                                                           L2B 417
R7I     INT     (QM4)+(QM6)=(QM4)                                               L2B 418
        GOT     (R3I)                                                           L2B 419
MI5     INT     (QM1)=(CP),(ZER)=(TE),(M8I)=(MSO)                               L2B 420
        GOT     (SOC)                                                           L2B 421
* PREPARE ML INFORMATION                                                        L2B 422
        DAT     0=ADC,0=MMC,0=STA,0=TMC,0=MMM,0=AP,0=BOC                        L2B 423
        DAT     M4M=M2M,M3M=M1M,TOP=ATP                                         L2B 424
PMM     INT     (ZER)=(ADC),(STA)'EQ'(ZER)                                      L2B 425
        IFF     (M4M)                                                           L2B 426
        INT     (M1M)=(MER),(BOC)=(ER)                                          L2B 427
        GOT     (ERR)                                                           L2B 428
M4M     INT     (ATP)+(STA)=(AP),((AP))'EQ'(EDM)                                L2B 429
        IFF     (M9M)                                                           L2B 430
M3M     GOT     ((MMM))                                                         L2B 431
M9M     INT     (ZER)=(MMC)                                                     L2B 432
M6M     INT     ((AP))=(TMC),(ONE)+(STA)=(STA),+(ATP)=(AP)                      L2B 433
M7M     INT     (TMC)=(TMM((MMC))),(ONE)+(MMC)=(MMC),(MMC)'GE'(TWO)             L2B 434
        IFF     (M6M)                                                           L2B 435
        INT     (TMM)'EQ'(ONE)                                                  L2B 436
        IFF     (M8M)                                                           L2B 437
        INT     (MMC)'LT'(SEV)                                                  L2B 438
        IFF     (M8M)                                                           L2B 439
        INT     (TAD((ADC)))=(TMC),(ONE)+(ADC)=(ADC)                            L2B 440
        GOT     (M7M)                                                           L2B 441
M8M     INT     (M2M)=(MML)                                                     L2B 442
        GOT     (PML)                                                           L2B 443
* PROCESSING BINARY OPERATION                                                   L2B 444
        DAT     0=KBP,0=SOP,0=DOP,0=RLC,0=SCM,0=IAT,0=LGC                       L2B 445
        DAT     TCM=ATC                                                         L2B 446
        DAT     R2B=M2B,R3B=M3B,R4B=M4B,R5B=M5B,R6B=M6B,R7B=M7B,SRC=M8B         L2B 447
        DAT     AT0=TA,AT0,AT0,AT0,AT0,AT5,AT6,AT0,AT0,AT9,ATU,ATV,AT0          L2B 448
        DAT     AT0,AT0,ATZ                                                     L2B 449
PB0     INT     (M2B)=(MSW)                                                     L2B 450
        GOT     (SWC)                                                           L2B 451
R2B     INT     (ZER)=(KBP),=(TCC)                                              L2B 452
B20     INT     (M3B)=(MSY)                                                     L2B 453
        GOT     (RSL)                                                           L2B 454
R3B     INT     (EM)'EQ'(ZER)                                                   L2B 455
        IFF     (B80)                                                           L2B 456
```

```
*********0+********0+********0+********0+********0+********0+********0+********0

        INT     (ATT)'GE'(LTY)                                          L2B 457
        IFF     (B3C)                                                   L2B 458
        INT     (ATT)-(LTY)=(IAT)                                       L2B 459
        GOT     ((TA((IAT))))                                           L2B 460
B3C     INT     (SYL)=(ER)                                              L2B 461
        GOT     (B04)                                                   L2B 462
AT0     LOG     (KBP)**(SIX),+(ATT)=(KBP)                               L2B 463
        GOT     (B20)                                                   L2B 464
AT5     INT     (YSL)=(SOP),(YDS)=(DOP)                                 L2B 465
        GOT     (B4C)                                                   L2B 466
AT6     INT     (YAS)=(SOP),(YDA)=(DOP)                                 L2B 467
B4C     INT     (M4B)=(MSY)                                             L2B 468
        GOT     (RSL)                                                   L2B 469
R4B     INT     (ATT)'EQ'(SOP)                                          L2B 470
        IFF     (B6C)                                                   L2B 471
        INT     (DOP)=(ATT)                                             L2B 472
        GOT     (AT0)                                                   L2B 473
B6C     LOG     (KBP)**(SIX),+(SOP)=(KBP)                               L2B 474
        GOT     (R3B)                                                   L2B 475
AT9     INT     (KBP)'EQ'(ZER)                                          L2B 476
        IFF     (B7C)                                                   L2B 477
        GOT     (AT0)                                                   L2B 478
B7C     INT     (ZER)=(SCM)                                             L2B 479
B8C     INT     (ZER)=(BOC)                                             L2B 480
B9C     INT     (TBO((BOC)))'EQ'(KBP)                                   L2B 481
        IFF     (B1C)                                                   L2B 482
        INT     (BOC)+(LGC)=(BOC),+(ATC)=(IAT),((IAT))=(STA)            L2B 483
        INT     (M5B)=(MMM)                                             L2B 484
        GOT     (PMM)                                                   L2B 485
R5B     INT     (EM)'EQ'(ZER)                                           L2B 486
        IFF     (SRC)                                                   L2B 487
        INT     (SCM)'EQ'(ZER)                                          L2B 488
        IFF     (R2B)                                                   L2B 489
        INT     (ZER)=(TDC),(YEQ)=(KBP)                                 L2B 490
        GOT     (B2C)                                                   L2B 491
B1C     INT     (ONE)+(BOC)=(BOC),'GE'(LBO)                             L2B 492
        IFF     (B9C)                                                   L2B 493
        INT     (KBP)=(ER)                                              L2B 494
B04     INT     (M8B)=(MER)                                             L2B 495
        GOT     (ERR)                                                   L2B 496
ATU     INT     (M6B)=(MOA)                                             L2B 497
        GOT     (POA)                                                   L2B 498
R6B     LOG     (KBP)**(ONE),+(CNE)=(KBP)                               L2B 499
        GOT     (B2C)                                                   L2B 500
ATV     INT     (ONE)=(SCM)                                             L2B 501
        GOT     (B8C)                                                   L2B 502
ATZ     INT     (M7B)=(MSY)                                             L2B 503
        GOT     (RSL)                                                   L2B 504
R7B     INT     (ZER)=(RLC)                                             L2B 505
B02     INT     (TRL((RLC)))'EQ'(SYL)                                   L2B 506
        IFF     (B03)                                                   L2B 507
        INT     (EQY((RLC)))=(ATT),(ZER)=(NCH)                          L2B 508
        GOT     (AT0)                                                   L2B 509
B03     INT     (TRL((RLC)))'EQ'(ZER)                                   L2B 510
        IFF     (B05)                                                   L2B 511
        INT     (SYL)=(ER)                                              L2B 512
        GOT     (B04)                                                   L2B 513
B05     INT     (ONE)+(RLC)=(RLC)                                       L2B 514
        GOT     (B02)                                                   L2B 515
```

```
*********0*********0*********0*********0*********0*********0*********0*********0
  * PROCESSING INTEG AND LOGIC                                          L2B 516
  PIT     INT     (ZER)=(GT),=(LGC)                                     L2B 517
          GOT     (PBO)                                                 L2B 518
  PLG     INT     (LBO)=(LGC),(ZER)=(GT)                                L2B 519
          GOT     (PBC)                                                 L2B 520
  * PROCESSING IFFALS                                                   L2B 521
          DAT     R2F=M2F,R3F=M3F,SRC=M4F                               L2B 522
  PIF     INT     (M2F)=(MSW)                                           L2B 523
          GOT     (WCS)                                                 L2B 524
  R2F     INT     (ZER)=(TDC),(M3F)=(MOA)                               L2B 525
          GOT     (POA)                                                 L2B 526
  R3F     INT     (ONE)=(GT),+(STA)=(STA),(M4F)=(MMM)                   L2B 527
          GOT     (PMM)                                                 L2B 528
  * PROCESSING GOTO AND MGOTO                                           L2B 529
          DAT     GNO=GN,GMN=GM,0=GTN                                   L2B 530
          DAT     G5T=G2T,G6T=G3T,SRC=G4T                               L2B 531
  PGT     INT     (GN)=(GTN)                                            L2B 532
  G7T     INT     (G2T)=(MSW)                                           L2B 533
          GOT     (WCS)                                                 L2B 534
  G5T     INT     (ZER)=(TDC),(G3T)=(MCA)                               L2B 535
          GOT     (POA)                                                 L2B 536
  G6T     INT     (ONE)=(GT),((GTN))=(STA),(G4T)=(MMM)                  L2B 537
          GOT     (PMM)                                                 L2B 538
  PGM     INT     (GM)=(GTN)                                            L2B 539
          GOT     (G7T)                                                 L2B 540
  * PROCESSING HALT                                                     L2B 541
          DAT     HNO=HTN,SRC=MHL                                       L2B 542
  PHT     INT     ((HTN))=(STA),(ZER)=(GT),(MHL)=(MMM)                  L2B 543
          GOT     (PMM)                                                 L2B 544
  * PROCESSING END                                                      L2B 545
          DAT     0=SCC,0=TSC,0=SC2,0=IDT,0=EC,$6J777777=ALL,0=SC3      L2B 546
          DAT     R2E=E2D,R3E=E3D,R4E=E4D,ED9=E5D                       L2B 5471
  PED     INT     (E2D)=(MSW)                                           L2B 548
          GOT     (WCS)                                                 L2B 549
  R2E     INT     (ZER)=(TDC),(E3D)=(MOA)                               L2B 550
          GOT     (POA)                                                 L2B 551
  R3E     INT     (ZER)=(GT),((GN))=(STA),(E4D)=(MMM)                   L2B 552
          GOT     (PMM)                                                 L2B 553
  R4E     INT     (ONE)=(DUP)                                           L2B 554
          MGC     $MOL                                                  L2B 555
          INT     (ZER)=(DUP),=(OB3)                                    L2B 556
  *               COMPARE SYMBOL TABLE WITH MISSING TABLE               L2B 557
          INT     (ZER)=(SCC)                                           L2B 558
  ED2     INT     (TMS((SCC)))'EQ'(ZER)                                 L2B 559
          IFF     (ED3)                                                 L2B 560
  ED9     INT     (TWO)+(SCC)=(SCC),'LT'(MSC)                           L2B 561
          IFF     (OER)                                                 L2B 562
          GOT     (ED2)                                                 L2B 563
  ED3     INT     (ZER)=(TSC)                                           L2B 564
  EE2     INT     (TSM((TSC)))'EQ'(TMS((SCC)))                          L2B 565
          IFF     (ED6)                                                 L2B 566
          INT     (ONE)+(TSC)=(TSC),(TMS((SCC)))=(IDT),(SCC)=(SC2)      L2B 567
  ED4     INT     (TMS((SC2)))'EQ'(IDT)                                 L2B 568
          IFF     (ED7)                                                 L2B 569
          INT     (ZER)=(TMS((SC2))),(ONE)+(SC2)=(SC3),(TMS((SC3)))=(NCR)  L2B 570
          INT     (TSM((TSC)))=(CBC)                                    L2B 571
          MGO     $SOU                                                  L2B 572
          GOT     (ED7)                                                 L2B 573
  ED6     INT     (TWO)+(TSC)=(TSC),'GE'(SMC)                           L2B 574
```

```
*********0*********0*********0*********0*********0*********0*********0*********0
         IFF    (EE2)                                                    L2B 575
         INT    (TMS((SCC)))=(ER),(E5D)=(MER)                            L2B 576
         GOT    (ERR)                                                    L2B 577
ED7      INT    (TWO)+(SC2)=(SC2),'GE'(MSC)                              L2B 578
         IFF    (ED4)                                                    L2B 579
         GOT    (ED9)                                                    L2B 580
OER      INT    (ALL)=(NCR),=(CBC)                                       L2B 5811
         MGO    $SOU                                                     L2B 582
         INT    (ONE)=(DUP)                                              L2B 583
         MGO    $SOU                                                     L2B 584
         INT    (ZER)=(DUP),=(EC)                                        L2B 585
OE2      INT    (EC)'LT'(ERC)                                            L2B 586
         IFF    (OE3)                                                    L2B 587
         INT    (TER((EC)))=(NCR),(EC)+(ONE)=(EC),(TER((EC)))=(CBC)      L2B 588
         INT    (EC)+(ONE)=(EC)                                          L2B 589
         MGO    $SOU                                                     L2B 590
         GOT    (OE2)                                                    L2B 591
OE3      INT    (ONE)=(DUP)                                              L2B 5921
         MGO    $SOU                                                     L2B 593
         HLT                                                             L2B 594
*                                                                        L2B 595
*                                                                        L2B 596
*              L2 COMPILER AUXILIARY PROGRAM                            L2B 597
* PRODUCING INDEX PART (MELCOM 1530)                                     L2B 609
         DAT    0=NAD,0=MCC,8175=OIX                                     L2B 610
         DAT    R2X=M2X                                                  L2B 611
PIX      INT    (ZER)=(CLC),=(TOM((CLC))),(ONE)+(OLC)=(OLC)              L2B 612
         INT    (OIX)=(TOM((OLC))),(ONE)+(OLC)=(OLC),(M2X)=(MUD)         L2B 613
         GOT    (PUD)                                                    L2B 614
R2X      INT    (THR)=(MCC)                                              L2B 615
M3X      INT    (TMM((MCC)))=(TOM((CLC))),(ONE)+(OLC)=(OLC)              L2B 616
         INT    (ONE)+(MCC)=(MCC),'GE'(SEV)                              L2B 617
         IFF    (M3X)                                                    L2B 618
         INT    (ONE)=(TOM((OLC))),(ONE)+(OLC)=(NAD),+(ONE)=(CLC)        L2B 619
         INT    (FOR)+(LC)=(LC),(ZER)=(QM3),=(QM4)                       L2B 620
         GOT    ((MIX))                                                  L2B 621
* PRODUCING INDIRECT ADDRESS (MELCOM 1530)                              L2B 622
         DAT    8172=OID                                                 L2B 623
         DAT    RID=I2D                                                  L2B 624
PID      INT    (ZER)=(TOM((OLC))),(ONE)+(OLC)=(OLC),(OID)=(TOM((OLC)))  L2B 625
         INT    (ONE)+(OLC)=(OLC),(I2D)=(MUD)                            L2B 626
         GOT    (PUD)                                                    L2B 627
RID      INT    (QM3)=(TOM((OLC))),(ONE)+(OLC)=(OLC),(ONE)+(LC)=(LC)     L2B 628
         INT    (QM4)=(TOM((OLC))),(ONE)+(OLC)=(OLC),(ZER)'LT'(QM5)      L2B 629
         IFF    (I3D)                                                    L2B 630
         INT    (LC)=(TOM((NAD)))                                        L2B 631
I4D      INT    (ONE)=(TOM((OLC))),(ONE)+(OLC)=(NAD),+(ONE)=(OLC)        L2B 632
         INT    (TWO)+(LC)=(LC)                                          L2B 633
         GOT    ((MID))                                                  L2B 634
I3D      INT    (ZER)=(QM3),=(QM4)                                       L2B 635
         GOT    (I4D)                                                    L2B 636
* PRODUCING OPERATION CODE AND BASE ADDRESS (MELCOM 1530)               L2B 637
POP      INT    (ZER)=(TOM((OLC))),(ONE)+(OLC)=(OLC),(QM1)=(TOM((OLC)))  L2B 638
         INT    (ONE)+(CLC)=(OLC),(QM3)=(TOM((OLC))),(ONE)+(OLC)=(OLC)   L2B 639
         INT    (ONE)+(LC)=(LC),(QM4)=(TOM((OLC))),(QM5)+(QM2),'EQ'(ZER) L2B 640
         IFF    (O5P)                                                    L2B 641
O4P      INT    (TMS((MSC)))+(ONE)=(TMS((MSC)))                          L2B 642
         GOT    ((MCP))                                                  L2B 643
O5P      INT    (LC)=(TOM((NAD)))                                        L2B 644
```

```
*********0*********0*********0*********0*********0*********0*********0*********0
          GOT     (O4P)                                                  L2B 645
    * GENERATING CONSTANT WORD                                           L2B 646
    GCW   INT     (SBT)'EQ'(ZER)                                         L2B 647
          IFF     (GW1)                                                  L2B 648
    GW2   GOT     ((MGC))                                                L2B 649
    GW1   INT     (ZER)-(SYL)=(SYL)                                      L2B 650
          GOT     (GW2)                                                  L2B 651
          END     (BEG)                                                  L2B 652
```

(2) $C^{L_{2,7090}}_{L_2} \longrightarrow L_{1530,1530}$ と $C^{L_{2,7090}}_{7090} \longrightarrow L_{1530,1530}$

および $C^{L_{2,7090}}_{L_{1530}} \longrightarrow L_{1530,1530}$ の対応

```
*********C*********O*********C*********O*********O*********O*********O*********O
        REG     TCO(50),TSM(1000),TMS(500),TER(200),TAD(15),TOM(20)          L28  44
        CDF     NCR(1),CBC(1),CAC(24),CB1(1),CB2(1),CB3(1),CUP(1)            L28  45
* INITIAL CONDITION SETTING                                                  L28  46
BEG     INT     (ZER)=(CCC),=(SPC),=(LCC),=(CDC),=(ERC),=(NCR),=(DUP)        L28  47
        INT     =(CBC),=(MSC),=(CSC),=(BLK),(UL)=(CMC),(CPW)=(BKC)           L28  48
BE1     LCG     (BLK)**(CHL),+(SPS)=(BLK)          PRODUCE BLANK WORD        L28  49
        INT     (BKC)-(ONE)=(BKC),'EQ'(ZER)                                  L28  50
        IFF     (BE1)                                                        L28  51
        GOT     (SRC)                                                        L28  52
* SUBROUTINES                                                                L28  53
* STACKING ERROR INFORMATIONS                                                L28  54
        CAT     0=ER,0=MER                                                   L28  551
ERR     INT     (ER)=(TER((ERC))),(ERC)+(ONE)=(ERC),(CDC)=(TER((ERC)))       L28  56
        INT     (ONE)+(ERC)=(ERC)                                            L28  57
        GOT     ((MER))                                                      L28  581
* READ ONE CHARACTER                                                         L28  59
        DAT     0=SIC,0=CRA,0=OCH,0=MRC                                      L28  60
        CAT     CAD=LIT                                                      L28  61
RCH     INT     (LIT)-(WDC)=(CRA),(CHC)+(ONE),'LT'(CPW)                      L28  62
        IFF     (R2F)                                                        L28  63
        INT     (CPW)-(ONE),-(CHC),*(CHL)=(SIC)                              L28  64
        LCG     ((CRA))//(SIC),*(EC1)=(CCH)                                  L28  65
        INT     (ONE)+(CHC)=(CHC)                                            L28  66
R3H     INT     (ONE)+(CCC)=(CCC)                                            L28  67
        GOT     ((MRC))                                                      L28  68
R2F     LCG     ((CRA))*(EC1)=(CCH)                                          L28  69
        INT     (ONE)+(WDC)=(WDC),(ZER)=(CHC)                                L28  70
        GOT     (R3F)                                                        L28  71
* GIVE ATTRIBUTE                                                            L28  72
* INTERNAL PATTERN TABLE (47 WORDS)                                          L28  73
        DAT     42=YOP,46,43                          (),.=+-/*$.'           L28  74
        DAT     41=YEQ,35=YPL,36=YMI,37=YSL,38=YAS,44=YDL,45,47              L28  75
        CAT     1,1,1,1,1,1,1,1,1,1                   NUMBER                 L28  76
        DAT     2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,0          L28  77
        DAT     0=GV1,0=ATT,0=MGA                                            L28  78
GVA     INT     (ZER)=(GV1)                                                  L28  79
GV3     INT     (TEX((GV1)))'EQ'(OCH)                                        L28  80
        IFF     (GV2)                                                        L28  81
        INT     (YOP((GV1)))=(ATT)                                           L28  82
        GOT     ((MGA))                                                      L28  83
GV2     INT     (ONE)+(GV1)=(GV1)                                            L28  84
        GOT     (GV3)                                                        L28  85
* READ SYLLABLE                                                             L28  86
        DAT     0=SYL,0=SLC,0=EP,0=SPC,0=LET,0=NCH,0=NAT,0=LAT,0=MSY         L28  87
        DAT     RR1=MR1,RR2=MR2                                              L28  88
RSL     INT     (ZER)=(SYL),=(EP),=(SLC),=(SPC),=(ATT),(ONE)=(LET)           L28  89
        INT     (NCH)'EQ'(ZER)                                               L28  90
        IFF     (RS2)                                                        L28  91
R2S     INT     (CCC)'LT'(C72)                                               L28  92
        IFF     (R3S)                                                        L28  93
        INT     (MR1)=(MRC)                                                  L28  94
        GOT     (RCH)                                                        L28  95
RR1     INT     (ATT)=(LAT),(MR2)=(MGA)                                      L28  96
        GOT     (GVA)                                                        L28  97
RR2     INT     (ATT)'EQ'(ZER)                                               L28  98
        IFF     (R7S)                                                        L28  99
        INT     (SLC)'EQ'(ZER)                                               L28 100
        IFF     (R3S)                                                        L28 101
        INT     (SPC)'EQ'(FIV)                                               L28 102
```

MAINCA

L2,7090 → L1530,1530
L7090

```
04070   0601 00 0 00416          STO     A00412
04071   0020 60 0 04054          TRA*    A04050
04072  +000000000000   A04066   OCT     000000000000
04073  +000000000000   A04067   OCT     000000000000
04074  +000000000000   A04070   OCT     000000000000
04075  +000000000000   A04071   OCT     000000000000
04076   0020 00 0 77457 A04072   TRA     A77775
04077   0500 00 0 04076 A04073   CLA     A04072
04100   0402 00 0 00407          SUB     A00403
04101   0601 00 0 04073          STO     A04067
04102   0500 00 0 00417          CLA     A00413
04103   0400 00 0 00370          ADD     A00364
04104   0402 00 0 00307          SUB     A00303
04105   0120 00 0 04131          TPL     A04125
04106   0500 00 0 00307          CLA     A00303
04107   0402 00 0 00370          SUB     A00364
04110   0402 00 0 00417          SUB     A00413
04111   0131 00 0 00000          XCA     0
04112   0200 00 0 00310          MPY     A00304
04113   0131 00 0 00000          XCA     0
04114   0601 00 0 04072          STO     A04066
04115  -0500 60 0 04073          CAL*    A04067
04116   0535 00 1 04072          LAC     A04066,1
04117   0771 00 1 00000          ARS     0,1
04120  -0320 00 0 00315          ANA     A00311
 4121   0602 00 0 04074          SLW     A04070
04122   0500 00 0 00370          CLA     A00364
04123   0400 00 0 00417          ADD     A00413
04124   0601 00 0 00417          STO     A00413
04125   0500 00 0 00370 A04121   CLA     A00364
04126   0400 00 0 00410          ADD     A00404
04127   0601 00 0 00410          STO     A00404
04130   0020 60 0 04075          TRA*    A04071
04131  -0500 60 0 04073 A04125   CAL*    A04067
04132  -0320 00 0 00315          ANA     A00311
04133   0602 00 0 04074          SLW     A04070
04134   0500 00 0 00370          CLA     A00364
04135   0400 00 0 00407          ADD     A00403
04136   0601 00 0 00407          STO     A00403
04137   0500 00 0 00367          CLA     A00363
04140   0601 00 0 00417          STO     A00413
04141   0020 00 0 04125          TRA     A04121
04142  +000000000052   A04136   OCT     000000000052
04143  +000000000056            OCT     000000000056
04144  +000000000053            OCT     000000000053
04145  +000000000051   A04141   OCT     000000000051
04146  +000000000043   A04142   OCT     000000000043
04147  +000000000044   A04143   OCT     000000000044
04150  +000000000045   A04144   OCT     000000000045
04151  +000000000046   A04145   OCT     000000000046
04152  +000000000054   A04146   OCT     000000000054
04153  +000000000055            OCT     000000000055
04154  +000000000057            OCT     000000000057
04155  +000000000001            OCT     000000000001
04156  +000000000001            OCT     000000000001
04157  +000000000001            OCT     000000000001
```

L2, 7092 → L1530, 1530
L1530

OBJECT PROGRAM

4013	000000016010	0	4103	000000017757	0	4173	000000016042	0
4014	000000000362	1	4104	000000003440	1	4174	000000000303	1
4015	000000016005	0	4105	000000000411	1	4175	000000016005	0
4016	000000000403	1	4106	000000004110	1	4176	000000004142	1
4017	000000016005	0	4107	000000016005	0	4177	000000017754	0
4020	000000000405	1	4110	000000000000	0	4200	000000004143	1
4021	000000016005	0	4111	000000016010	0	4201	000000004203	1
4022	000000000406	1	4112	000000000411	1	4202	000000016010	0
4023	000000016005	0	4113	000000016022	0	4203	000000000000	0
4024	000000000404	1	4114	000000000363	1	4204	000000016305	0
4025	000000016005	0	4115	000000016005	0	4205	000000004142	1
4026	000000000411	1	4116	000000000411	1	4206	000000016013	0
4027	000000016005	0	4117	000000016010	0	4207	000000000310	1
4030	000000015755	2	4120	000000000404	1	4210	000000000053	0
4031	000000016005	0	4121	000000017757	0	4211	000000000053	0
4032	000000015720	2	4122	000000003440	1	4212	000000016005	0
4033	000000016005	0	4123	000000000411	1	4213	000000004144	1
4034	000000015754	2	4124	000000004126	1	4214	000000016010	0
4035	000000016005	0	4125	000000016005	0	4215	000000000363	1
4036	000000000407	1	4126	000000000000	0	4216	000000016022	0
4037	000000016005	0	4127	000000016010	0	4217	000000000412	1
4040	000000000401	1	4130	000000000363	1	4220	000000016005	0
4041	000000016005	0	4131	000000016022	0	4221	000000000412	1
4042	000000000377	1	4132	000000000411	1	4222	000000016010	0
4043	000000016010	0	4133	000000016005	0	4223	000000000363	1
4044	000000000307	1	4134	000000000411	1	4224	000000016022	0
4045	000000016005	0	4135	000000017754	0	4225	000000000403	1
4046	000000000410	1	4136	000000004100	1	4226	000000016005	0
4047	000000016010	0	4137	000000004141	1	4227	000000000403	1
4050	000000000302	1	4140	000000015756	0	4230	000000017754	0
4051	000000016005	0	4141	000000000000	0	4231	000000004145	1
4052	000000000400	1	4142	000000000000	0	4232	000000004234	1
4053	000000016010	0	4143	000000000000	0	4233	000000015756	0
4054	000000000377	1	4144	000000000000	0	4234	000000000000	0
4055	000000016270	0	4145	000000000000	0	4235	000000017754	0
4056	000000000303	1	4146	000000015753	2	4236	000000004143	1
4057	000000016022	0	4147	000000016010	0	4237	000000004241	1
4060	000000000057	1	4150	000000004146	1	4240	000000016010	0
4061	000000016005	0	4151	000000016032	0	4241	000000000000	0
4062	000000000377	1	4152	000000000402	1	4242	000000016013	0
4063	000000016010	0	4153	000000016005	0	4243	000000000310	1
4064	000000000400	1	4154	000000004143	1	4244	000000000053	0
4065	000000016032	0	4155	000000016010	0	4245	000000000053	0
4066	000000000363	1	4156	000000000412	1	4246	000000016005	0
4067	000000016005	0	4157	000000016022	0	4247	000000004144	1
4070	000000000400	1	4160	000000000363	1	4250	000000016010	0
4071	000000016032	0	4161	000000016032	0	4251	000000000363	1
4072	000000000362	1	4162	000000000302	1	4252	000000016022	0
4073	000000016170	0	4163	000000016177	0	4253	000000000402	1
4074	000000004053	1	4164	000000000000	0	4254	000000016005	0
4075	000000015756	0	4165	000000016010	0	4255	000000000402	1
4076	000000000000	0	4166	000000000302	1	4256	000000016010	0
4077	000000000000	0	4167	000000016032	0	4257	000000000362	1
4100	000000000000	0	4170	000000000363	1	4260	000000016005	0
4101	000000016010	0	4171	000000016032	0	4261	000000000412	1
4102	000000004077	1	4172	000000000412	1	4262	000000015756	0

〔第Ⅱ編　参考文献〕

(1) Halstead, M·H·:〝Machine-independent Computer programming〟, Spartan Books（1962）

(2) 藜野：〝Compilcr Generating System〟, 第6回プログラミング・シンポジウム報告集，B－21（昭40－01）

(3) 井上：〝ALGOL言語によるALGOLコンパイラの製作〟, 第6回プログラミング・シンポジウム報告集，B－44（昭40－01）

(4) 首藤，漁田，居原田：〝計算機基本言語に関する　一考察〟, 三菱電機技報，39－3, P.431（昭40－03）

(5) 首藤，魚田，居原田：〝計算機を用いたコンパイラ作成自動化の実験〟, 第7回プログラミング・シンポジウム報告集，C－53（昭41－01）

(6) Sudo, Uota, Iharada：〝Some Considerations and Experiments on the Basic Programming Language〟, MITSUBISHI DENKI LABORATORY REPORTS, 7－1, P.40（1966－01）

(7) 首藤，魚田：〝コンパイラ自動作成の一方法〟, 信学誌，50－4，P.649（昭42－04）

第Ⅲ編　オンライン制御用計算機における言語処理システム

第1章　緒　　論

プロセス制御を中心とするオンライン制御は計算機応用の一つの大きな分野であり，早くから実用化の努力がなされて来た。プロセス制御応用におけるプログラミング言語の研究も汎用の分野でFORTRANおよびALGOLが実用化されて間もなく試みられている。しかしながらプロセス制御応用がオンラインシステム構成に関する問題を本質的に含んでいるために，標準的な汎用言語を流用することはほとんど成功しなかつた。初期の汎用言語は，オンライン処理を含まない処理形態を基本として作られ，そのプログラムの実行の制御を受持つシステムも今日のオペレーテイングシステムからみると単純で機能の乏しいものであつたから，その面の充実を待たなくては制御用のプログラミング言語システムとして一貫した形で実用に供することはできなかつたのである。
また，入出力機能についても制御用の場合は特殊なものを要求し，初期の汎用言語で考えられていた機能では到底不足であり大幅な拡充を要したことも一因であろう。その他に，高レベル言語を使用するために要求されるハードウエアの追加（コンパイラ言語を使うためにはアセンブラ言語による場合よりもコアメモリ4 kW を余分に要する，といつたもの）のための投資がプログラム作成の人件費よりも重大視された事情が大きな要因となつている。これらの結果として，汎用の分野で標準言語が普及したにもかゝわらず，最近まで制御用の分野ではアセンブラ言語が主要なものとして使用されて来たのである。

汎用の面でオンライン処理が次第に意識され，またプログラミング言語とコンパイラだけでなく実行の制御を司るオペレーテイングシステムが概念上も整理され機能上も充実したものが実用化されるに至り，またプログラム開発の人件費の圧力が高まるにつれて，制御用言語の認識が高まり，言語と処理システムが開発され始めた。(1) ～ (5)

プロセス制御応用におけるプログラミングシステムに対して要求される事項を整理すると次のようになる。[6], [8], [9], [14], [15]

(1) プログラム作成の時間と労力の節減。

これは汎用と変りない。アセンブラ言語を脱して高レベルな言語を使用する要求である。

(2) 制御に必要な機能をもち，実行時の時間効率およびメモリ等の利用効率が高いこと。

これは作成方針および技術の問題である。汎用と比べて制御用の場合はタイミング上の要求が厳しいから，コンパイル処理に時間をかけても実行時効率の高いオブジエクトプログラムを生成することが要求される。

(3) 制御を専門とする技術者が使いやすいもので，計算機側との情報伝達が確実であること。

これは言語の表現が簡単明快であり，制御用で使う情報の表現形式とできるだけ近いものが要求されることを示す。

(4) ドキュメンテーションの合理性。

プログラミング言語システムに対する要求として，プログラム論理の表現のほかに，一たん作つたプログラムを長期間実用に供するためにその内容の記述，他との関連，修正の記録，それらの更新といつた側面的な機能が極めて重要である。特に多人数でシステムを仕上げるとき，あるいは運用担当者の変更が予想されるときにはこの問題が大きな意味をもつ。

(5) 現地での据付調整に伴うプログラムの修正の容易さ。

これは制御用に特有であろうが，プログラム中のパラメータの一部がプラント試運転の結果をみて最終決定されたり，時には論理に対する修正要求が出る。また，現地設置の計算機の構成は工場の計算センターの機械に比べ種々不自由なことが通例である。これらに対処するために補助的なプログラムシステムの機能が要求される。

制御用の言語を開発するにあたつて，上記の諸要求をいかにみたすかが重要な問題となるが，アセンブラ言語より高位の言語レベルとして次の二つがあり

それぞれに特色がみられる。

(1) 手続き向き言語

これはFORTRAN，ALGOL と同レベルのいわゆるコンパイラ言語で，制御用の機能を適当な形でまとめることにより作り出される。これは制御用の分野では一応汎用的に使われる。

(2) 問題向き言語

手続き向き言語よりもつと制御処理操作に密接な対応をもつ表現を用い，既製のパッケージプログラムと組合せてプログラムを組立てようとするものである。使い方が著しく容易になる反面，適用対象が制限される。

制御用として充分な機能をもつことがこれらの言語に共通に要求されることは当然であるが，両者の併用，更に特殊部分に対処するためアセンブラ言語との併用が望まれ，あるいは必要となるから，各々の言語のオブジエクトプログラムがアセンブラ言語のそれと等質構造をもつことが要求される。

本編では，制御用の手続き向き言語および問題向き言語システムについてその構成上の問題点とその対処法を実例を用いて記述し，最後に補助機能の一つとしてオンライン計算機におけるプログラムシミュレーションにふれる。

第2章　科学技術計算用言語の制御用への拡張

2.1　緒　　言

プロセス制御応用に於ては，計算機で行なう処理の形態が科学技術計算と類似しているためにその標準言語であるFORTRAN，ALGOL を流用する試みがみられている。しかし制御応用には，2進情報の積極的使用，多重タスク制御，特殊な入出力処理など特有の情報形態，処理操作が必要であり，上述の標準言語ではこれらに対する配慮がないため，サブルーチン群の使用，部分的にアセンブラ言語を使用などの方法で対処することを与儀なくされている。その結果としてオブジエクトプログラムの時間的，空間的な効率の低下あるいは標準言語をそのまゝ用いられないためのプログラム作成能率の低下という問題を残している。

これを解決するために，標準的な科学技術計算用言語の機能を拡張して制御応用における要求を満たす言語を形成する方法が考えられる。FORTRAN を母体として制御用の手続き向き言語を構成する場合の，必要機能，命令文の構成，コンパイラ構成上の問題について述べる。

2.2　制御用計算機に要求される機能

FORTRANを母体としてプロセス制御用の手続き向き言語を構成するときに，拡張を要求される機能は次のようにまとめられる。(6)

(1)　情　報　の　形

2進データの記述と処理の機能が要求される。

制御応用で使う2進データは，リレー接点，スイツチの状態に代表されるもので，単純であるが数が多く，それらを組織立てて集めておき，走査あるいはセツト，リセツトすることが必要となる。これはFORTRANの論理処理機能では冗長かつ低効率で使用に耐えないし，汎用のPL/Iのもつビツト列処理機能を以つてもなお問題を残す。(8)

基本的に2進のデータ形をもち，それを基にした構造データ，および

2進処理を融合させた一連の処理命令が必要である。なお，処理速度を上げるために語構成と2進データの関係を幾らかハードウエア指向的に表現することは止むを得ぬこととされよう。

(2) スーパーバイザの機能

最近の計算機ではマルチプログラミングの機能を備えたオペレーティングシステムをもつことが普通になつてきている。制御応用ではプラントのもつ本質的な多重性に対応して制御操作を行うために，系の多重性にプログラムの多重性を対応づけて計算機を働かせることができると好都合である。初期の制御計算機ではこれを定時間隔走査で逃げてきたともいえる。

この場合，プログラムの比較的小さなセグメントが単位となる多重処理が適するから，スーパーバイザのタスク制御機能がそのまゝ使えるとよい場合が多い。このタスク制御機能は通常アセンブラ言語の機能として，マクロサブルーチンなどの形で実現されているから，これを手続き向き言語の文として直接使えるよう，言語機能の拡張が要求される。

(3) 外部メモリの使用

汎用の計算機では外部メモリをデータファイルとして扱い，入出力の対象としてきた。（最近IBM−370で仮想メモリの概念が使われ，この考え方がようやく変ろうとしている）。制御用計算機では外部メモリを含めて一つの世界を構成した使い方が殆どで，外部メモリは入出力の対象のファイルとしてではなくコアメモリの延長のデータあるいはプログラムの置き場とみることが通例である。しかも外部メモリとの転送を含めて厳しいタイミング条件におかれているから，直接的なアクセス法と命令文が必要となる。FORTRANにはこの機能は無いから拡張が必要である。

(4) 入　出　力

プロセス入出力機能が要求される。

アナログ，デイジタルの各種のプロセス入出力機能をなるべく汎用的な形で備えることが必要である。

(5) プログラム構造

制御用プログラムは，一つのプラントに対して全体として制御設備のプログラム部分という形をとる。これは，計算機の中をプログラム処理作業が流れて通つて行く形をとる汎用計算機の場合との大きな相違点である。

制御用プログラムは，多くのセグメントあるいはモジユールが有機的に組合わされることが前提であるから，モジユール結合に際して構造の指定を積極的にできること，および各モジユール間の共通データも構造の規定に関連して指定でき共用できることが言語の機能として要求される。

(6) 他言語との混用性

ここでは特殊な処理をアセンブラ言語でプログラムして，それをこの言語の途中に任意に挿入できることを指す。セグメント単位での混用についてはオペレーテイングシステムに備えられる編集機能により大して問題なく実現できよう。

2.3 制御用手続き向き言語の構成

前節に掲げた機能をFORTRANの機能要素に組込む方法について，本節で具体的に述べる。例として，中型の制御用計算機MELCOM−350／30のために作られたCONFORM (Basic Translator for Industrial CONtrol FORMula) の表現を用いる。

2.3.1 2進形データの記述と処理

(1) 記　　述

FORTRANにはデータの表わし方として，定数，変数，配列，および配列要素があり，各々に整数形と実数形がある。

CONFORMではデータの数値の形として2進形を設けてビット集合（ここではハードウエアの語構成と合わせて16ビットとする）を表わすほか，2進因子という表わし方で1ビットデータを扱う。

2進形の定数は16進数で

"08AF"

のように示す。

変数，配列，および関数に対する形宣言文に2進形宣言BINARYを追加する。

2進因子は2進形データで示される16ビットの中の1ビットを指定するもので

変数に対しては　　$v("j)$

配列要素に対しては　　$v(i_1,\cdots,i_n,"j)$　　$(n\leq 3)$

で表わす。2進因子のビット位置関係を図2.1に示す。

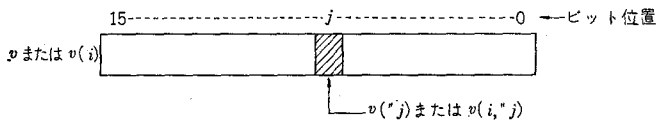


図2.1　2進因子の指示ビット

(2)　2進代入文

算術代入文に加えて2進代入文を設ける。これは2進データについての論理演算を行うもので次の形の一般形をもつ。

$v = e$

e は2進論理式である。

2進論理式は二つの2進データについての論理演算を記述するもので，一般形は次の通りである。

v_1 または $v_1 \phi v_2$

v_i は2進形データか2進因子のいずれかで，ϕ は論理演算記号を表わし，.AND.（論理積），.OR.（論理和），が指定できる。各々の v_i の前に.NOT.を付加して否定の値（1と0を逆にした値）を演算の対象にすることもできる。

演算は，v_1，v_2 が2進因子の場合，1ビットのみが，また，2進形データの場合，16ビットが並列に行われる。

2進代入文の例としては次のようなものがある。

A＝B．AND．C(I，J)

D＝．NOT．B．OR．C(I−2，5)

Z(＂I)=B(＂I)．AND．．NOT．C(I，J，＂I)

(3) 実行制御文

GOTO文，IF文，DO文の変形として次のものを追加する。

a) 2進 IF 文

IF v k_1，k_2

2進因子 v が0なら k_1 の文へ，1ならば k_2 の文へ行く。

b) 2進関係 IF 文

IF(e) k_1，k_2

2進関係式 e が成立するならば k_1 の文へ，不成立ならば k_2 の文へ行く。e は，二つの2進形データか2進因子を．EQ．(等)または．NE．(不等)で結んで表わす。

例 IF(X(＂I)．NE．．NOT．Z(＂J) 10，11

(4) ビット処理文

ビットデータの処理は2進代行文でも記述できるが，さらに機能化されたビット処理用の文を設ける。

a) SETBIT 文

SETBIT v

RESETBIT v

2進因子 v で示されるビットを1（SETBIT）または0（RESETBIT)にする。

b) SCAN文

SCAN v，l，p，k

2進因子 v から始まる l ビット領域（$1 \leq l \leq 16$）中の最初の1のビット位置を整数変数 p に入れ，文番号 k の文を次に実行する（図2.2）。

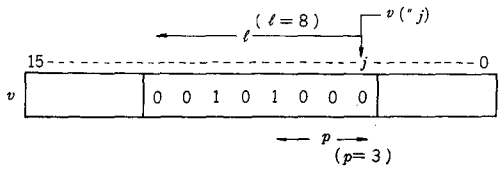


図 2.2　SCAN文の機能概念

c)　DOSCAN 文

DO−SCAN　v , l , p , k

2進因子 v から始まる l ビット領域（$1 \leq l \leq 2^{15}-1$）中のビットを調べ，1のビットに遭遇するごとにその位置を p に入れ，DO−SCAN文から文番号 k の範囲の文を実行する（図 2.3 ）。

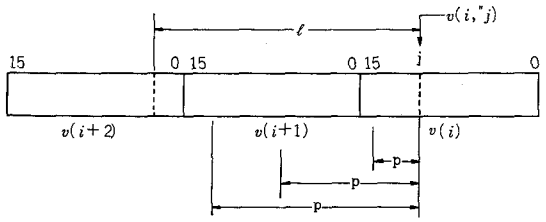


図 2.3　DO−SCAN文の機能概念

2.3.2　タスク制御文

タスクの多重処理はタスク制御文を設けることにより，簡潔にしかも他の文との一様性をそこなうことなく記述できる。

(1)　ASSIGN−TASK 文

ASSIGN−TASK　t (a_1 , a_2 , … , a_n)

タスク名の集団 a_1 , a_2 , … , a_n に代表タスク名を定義する。要素のタスク名は $t(i)$ の形式で表わす。

(2)　CALL−TASK 文

CALL−TASK　t , p , z , c , u

タスク t に優先順位 p（整数）を与え，z で指定するタイミングで起動する。z が ECB のとき即時起動，TIME とすれば指定時刻に，INTERVAL とすれば指定時間後に，それぞれ起動する。時間

Cは，分とシステムクロックの整数倍を単位として示す。uはユーザーの使用にまかされる番号である。

(3) WAIT文

WAIT t, ECB w, k

タスクtの終了同期をとる。kはタスクtにエラーが発生したときの行先番号である。

(4) PURGE 文

PURGE t, u

ユーザー番号uのタスクtを強制的に消滅させる。

(5) DELAY 文

DELAY c

タスクを時間cだけ休止させる。

図 2.4 はタスク制御文の例であり，簡単なタスクスケジュールプログラムを示している。この例では，タスクの起動要求はコア常駐ワード内の対応するビットが1であることにより示されているものと仮定している。全ビットを調べ終ると，1分30秒後経過に再び同様の操作をくり返す。

```
C FOR COMM. STATE. NO.   CONT   CONFORM STATEMENT
LOCATION                 OPERATION   OPERAND   REMARKS
FOR ASSEMBLER CODING
C  TASK  SCHEDULE PROGRAM
         MAIN  SCHEDL
         BINARY GLOBAL (SCHTAB)
         ASSIGN-TASK TSET(TASK1,TASK2,TASK3,TASK4
       1                 ,TASK5,TASK6,TASK7,TASK8)
     55  DO-SCAN SCHTAB("0),8,I,33
         RESETBIT SCHTAB("I)
         I=I+1
     33  CALL-TASK TSET(I),1000,ECB 0,1
         DELAY (1,120)
         GO TO 55
         END
```

図 2.4

2.3.3　外部メモリと内部コアメモリとの直接転送

MELCOM−350／30 には高速大容量のドラムが標準装備されている。前述のようにこれを内部メモリの領域の延長として扱うために，両者間のデータ転送用としてFETCH文，STORE 文を追加する。

FETCH (v_1 , v_2 , v_3), (v_1', v_2', v_3'),…

STORE (v_1 , v_2 , v_3), (v_1', v_2', v_3'),…

v_1 で内部メモリ領域の配列を指定し，v_2 で外部メモリ領域のアドレスを直接指定する。機能はFETCH 文で外部メモリから内部コアメモリへ，STORE 文でその逆の転送を行なう。

例　FETCH (A(1),"28FF",160),(B(1),IDA,$\overline{\text{NO}}$)

STORE (D,4096,64)

2.3.4　プロセス入出力

プロセス入出力には，デイジタル入出力，アナログ入力，パルスバッファ入力，パルス幅出力があるが，これらを一括して次の文で扱う。

例　PROCESS−I／O(i , p) v_1 ,…, v_n　　($n \leq 4$)

i , p で入出力動作の種類，リレー番号，ゲインなどを所定のビットパターン（２ワード）で指定する。v_i は入出力の対象となるプロセス変数である。

PROCESS−I／O("C800",PRCW)AN1,AN2

2.3.5　サブプログラムの結合構造指定

サブプログラムの結合のしかたを指定するためにSTRUCTURE 文を設ける。

主プログラムとサブプログラムの構造上の結合形式は，サブプログラム側に個有の特性として持たされているのではなく，主プログラム側での引用時にどのような形に結合するかをSTRUCTURE 文により指定する。

STRUCTURE z_1 (s_1 ,…, s_n) , z_2 (s_1',…, s_n'),…,

z_i は主プログラム内で呼ばれているサブルーチン s_i の結合構造を規定するもので，PHASE（フエーズオーバーレイ），SUB （フエ

ーズ内常駐），LOCAL（フエーズ内オーバーレイ），GLOBAL（コア常駐）の指定が可能である。

2.3.6 アセンブラ言語との混用

CONFORM言語のプログラムの途中にアセンブラ言語の文を自由に書くことを許し，コンパイラレベルの言語機能では直接書きにくい特殊な部分をアセンブラ言語で書き，全体としてコンパイラレベルの言語で効率よくプログラミング作業を進めることを可能とする。

アセンブラ言語への切換えには

```
++ASSEMBLER
```

を，CONFORMへの切換えには

```
++CONFORM
```

を用いる。

両言語間での名前の相互参照が必要であるが，コンパイラ側のものはアセンブラ側で自由に使用でき，アセンブラ側の名前は形宣言文でRESERVED宣言をしてコンパイラ側で使用する。例を図2.5にあげる。

FOR ASSEMBLER CODING

C FOR COMM. STATE. NO.	CONT.	CONFORM STATEMENT
LOCATION		OPERATION / OPERAND / REMARKS
		BINARY (A)
		REAL RESERVED (TITLE(12))
		:
		GO TO 200
		:
++		ASSEMBLER
S$0200		LL A LOAD A
		SRR 8 RIGHT ROTATE 8 BITS
		STL A STORE IN A
		B S$0300
TITLE		DC W"PROCESS CONTROL COMPILER"
++		CONFORM
		:
300		WRITE (2,50) TITLE
50		FORMAT (H"1",6A4)
		:

図 2.5

2.4 CONFORM コンパイラ

2.4.1 実行時プログラムの構造

プロセス制御用のプログラミングシステムでは，プロセス制御特有の処理機能を記述する能力とともに，コンパイラにより生成される実行（オブジエクト）プログラムの占有メモリ，実行速度など，プログラムの構造面における効率の良さが強く要求される。ことに，コンパイラレベルの言語で書かれるプログラムは制御用プログラム全体をまかなうものでなく，あくまで，管理プログラムおよび既製のパッケージを含めたプログラム全体の構成部分として準備される。従つてシステム全体が一体構造で動作できるためには，コンパイラレベルで準備されたプログラムとシステムの他の要素とのつながり具合が一様でなければならない。

このことから，オブジエクトプログラムの構造設計上目標となる事項は次の二項であるといえよう。

(1) プログラムの占有メモリを少くする。

(2) アセンブラレベルのきめ細かさで，他のプログラムとの接続ができること。

MELCOM−350／30 CONFORMでオブジエクトプログラム構造設計上具体的に配慮した事項は次に列記する通りである。

(1) 部分的ルーチンの生成法

比較的まとまつた仕事をするルーチン部分はサブルーチン形式で生成することが原則となるが，画一的な処理をさけて，タスク制御文のように直接挿入した方が効率の良いものは，適宜，埋込式を採用する。

(2) 添字計算

添字は，配列名に関係なく添字の形ごとに定数表を作成し，添字計算用サブルーチンにこの表をパラメータとして与えて計算する。この方法では配列名を問わず同形の添字が多く現われるほどメモリ

節約に効果がある。

(3) FORMAT 制御付き入出力

MELCOM−350／30 では周辺機器を対象とする入出力は，データの編集と変換の機能を持つたI／Oリーダ・ライタと呼ばれるサブシステムにより集中管理される。CONFORMでもこれを利用してFORMAT 制御付き入出力ではすべてこのサブシステムに接続する処置をとり，それらの部分に対するオブジエクト列を極めて簡単なものにしている。

(4) 外部メモリ入出力

外部メモリを内部コアメモリの延長領域とみなして少しでも効率よく使用する意図で外部メモリ入出力命令を設けたのであるから，これらの部分に対するオブジエクト列は，直接スーパーバイザがもつデータ転送機能を借りて使うように生成する。

(5) プロセス入出力

プロセスデータは応用システムごとにコードの種類，構成が異るので，編集や変換機能まで提供すると汎用性の面で問題がある。したがつてプロセス入出力についてもスーパーバイザ機能を直接利用する形でオブジエクト列を生成する。

(6) サブルーチンの構造およびオペレーテイングシステムとのつながり

サブルーチンの結合構造の選択は従来アセンブラレベルのみで可能であつたが，CONFORMでは前述の言語機能に従つた結合構造によりオブジエクト列を生成できる。その他，コア常駐データやアセンブラ言語で書かれたプログラムとのシンボルの結合，別の標準形FORTRANのサブルーチンとの互換性など，他のプログラム要素とのきめ細かいつながりを保つために種々配慮した。

2.4.2 コンパイラ

CONFORMの言語はFORTRANを母体としており，拡張機能の表現

にもFORTRANのもつ標準的な形をできる限り踏襲している。またオブジエクトプログラムに現われる拡張機能部分については前述のようにスーパーバイザの機能あるいはサブシステムにより裏付けがある。従つて，CONFORMコンパイラは特別な方法によらずに構成することができる。

初版のコンパイラでは，アセンブラ言語との混用を簡単に実現するために，オブジエクトプログラム言語としてアセンブラ言語を採用する間接的な方式を用いた。これにより，アセンブラ言語の全機能をCONFORMと併用することを可能とし，また，コンパイラとアセンブラがジヨブコントロールプログラムを介して別タスクとして起動，実行されるためにオンラインコンパイルを有利にする作用が得られた。反面コンパイル速度は直接的な方式に比べて低下するが，プログラムの実行に比してコンパイルの頻度が低いため実行時効率が重点とされる制御用であるために救われた形となつている。

その後この言語およびコンパイラは更に広い用途に適用するため，FORTRANの標準文法を包含させるとともに，オブジエクトプログラムの最適化を実施するものに改訂された。これをCONFORM－Ⅳ と呼んでいるが，ここではコンパイルに直接方式を用い，それと共にコンパイラ言語と混用できるアセンブラ言語の機能を必要なものに制限する処置をとつた。

2.5 結　　言

標準的な科学技術計算用言語を母体として機能拡張によりプロセス制御用の手続き向き言語を構成する問題について，特に要求される機能と，FORTRANを母体として開発したCONFORMにおける実現方法を中心に述べた。

この種の言語は制御用の分野では汎用性がある。そのためにFORTRAN等の汎用計算機の言語にならつて標準化の必要が指摘されている[7),(8),(17),(18)]。その場合，2進情報処理機能，タスク制御機能のようにFORTRANの文法からか

なり離れた性質のものをFORTRANのもとの文法と融和し，かつ機種個有の特性に依存させぬ形で標準形にまとめることはかなり困難であり，むしろそのような機能をある程度考慮して作られたPL／Iを母体とする方が無理なく標準化を進め得ると思われる。ただし，PL／Iは大型の計算機ではじめて意味をもつ多種の機能を備えているので，中型以下が主対象となる制御用では実用的な言語を作るにはPL／Iの機能をかなり大幅に削減することが必要である。

第3章　問題向き言語システムの構成(22)～(25)

3.1　緒　　言

プロセス制御計算機の応用では，対象に応じてその制御処理が多様である反面，入出力に付帯する処理，制御計算の中に現われる手法については定形的なものも多く含まれている。この定形的な処理に対して専用のサブシステムあるいはプログラムパッケージを用意しておき，制御用プログラムの構成にあたつて，それらを必要に応じて集成し，あるいはパラメータを与えて，一つのプログラムに組み立てる方法をとると，プログラム作成の効率化に効果的である。このとき，パッケージの指定，パラメータの指定，相互の接続関係および優先順位の指定などを一括して系統的に，行なえるようシステム化しておくと，それらの指定のしかたを一つの言語とみて，それでプログラムを書くように使用することができる。この言語は，FORTRANなどの言語に比して制御対象に密着する度合が強いもので，問題向き言語と呼ばれる。

この種の原語は適用対象がFORTRANなどの手続き向き言語に比して限定される欠点があるが，制御対象に密接に対応した簡潔な表現で成つているので，プログラム作成の省力化，特にプラント技術者にプログラム作成をさせる場合の作業の簡易化，プログラムに伴つて必要となるドキュメンテーションの合理化に効果があり，最近注目されつゝある。

ここではDDC（直接計算制御）およびSCC（監督制御）を主対象とした制御用の問題向き言語システムについて，システムの構成，プロセッサの概要，制御アルゴリズムのパッケージ化などを実例にふれつゝ述べる。

3.2　制御用問題向き言語システムの構成

3.2.1　プログラムの基本形態

プロセス制御における計算機の処理機能には種々なものが含まれる

が，基本的には図 3.1 に示すように，プロセス入出力処理，制御計算処理に分けられ，それら相互の仲介をデータファイルによつて行なう形に整理することができる。

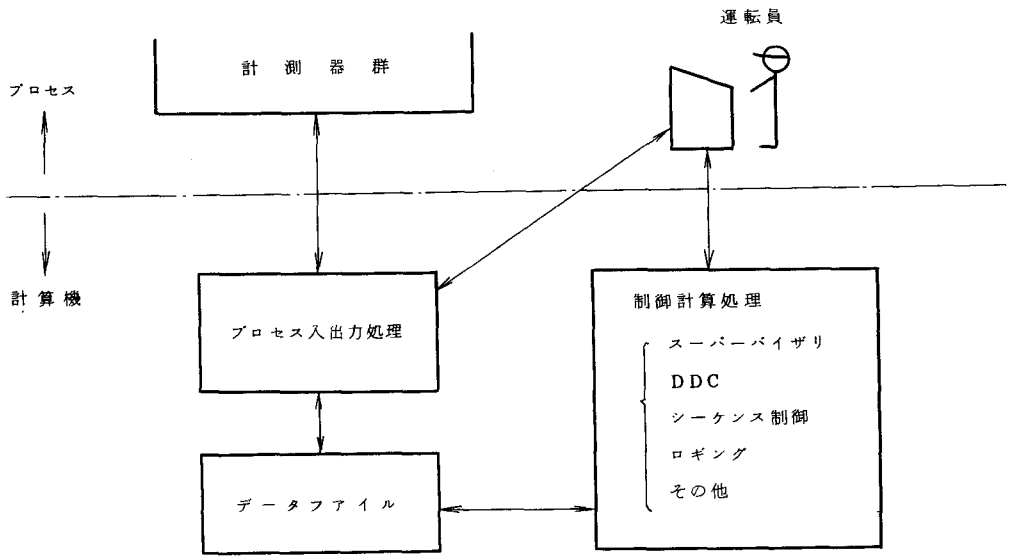


図 3.1

プロセス入出力処理には，入力機能として，プロセスデータを入力し，制御計算処理で使用できる形式に変換し，各種の限界値と比較して妥当性を確認した上で，所定のデータファイルに格納する機能があり，出力機能として，制御計算の結果得られて出力ファイルに格納されているデータ要求に応じて，プロセス側へ出力する機能がある。一つのプラントを制御するシステム全体としては，これら入出力の点数が多く，また各種のデータのやりとりは，周期的にデータ点をスキャンする操作を伴なうか，あるいは各種のイベントの連鎖に基づいて組織的に行なわれる。これらの操作は，個々には複雑な制御を伴うが全体として外見上は共通的な処理から成ると見なし得るため，このプロ

セス入出力部分を制御計算部分と分離して，サブシステムとしてまとめると全体のプログラムシステム構成が簡明になつて，好都合なことが多い。

制御計算処理機能部分では基本的には，プロセス入力処理によつて変換格納されたデータを使用して，制御のアルゴリズムに応じた各種の計算を行ない，プロセスへの制御出力データを算出して，出力機能部分へ渡す機能をもつ。この処理は適用対象および制御の方式によつて多様性に富んでおり，適当なプログラミング言語を用いて個々に書かれる部分である。

この様に入出力と内部処理を機能部分として分離し，全体をタイミングメカニズムあるいはイベントの連鎖によつて管理する形をとることが，制御用計算機プログラムの特徴であり，計算手続きと入出力とが混在し，一体化した形をとる汎用のプログラム構造と大きく相異するところである。勿論すべてのプロセス制御プログラムが，計算手続きと入出力の明確な分離を前提としているわけではなく，メッセージ出力その他ローカルな制御と外界との情報リンクに伴う入出力処理は計算手続きに含まれるし，プラントへの出力も手続きプログラムで処理されることもある。

3. 2. 2　プログラミング言語の定形化

プロセス入出力処理機能は，前述のように共通的な処理が多いため，各々のデータごとに，記号名，周期，印字形式，装置（接点など）のアドレス，工学単位，入力値から工学値への変換方式，上限・下限の値，目標値，初期値，代替装置，積算，平均などの処理指定などを記入指定する書式を定めることによつて，ほぼ定形的に処理の指定を行なうことが可能である。これらの指定は穴埋め（fill-in-the-blahk）方式による入出力処理プログラムとみなすことができる。この穴埋め方式の書式はFORTRAN等の手続き向き言語の入出力命令より，はるかに対象に密着した表現を用い得るため，非専門家によるプログラミ

ングの簡易化に役立つとともに，書かれたプログラムをそのまゝ処理手続書とみなすことができて，ドキユメンテーションの合理化にも効果的である。

制御計算処理機能については，入出力部分ほど定形化の必然性がない。そのため，入出力部分のみを定形化した言語で書き，制御計算については，アセンブラ言語あるいは拡張形FORTRANで書く混成法が可能であり，現実にそれを行なうシステムの例もある。しかし，プログラミング言語の定形化を進める程，プログラム作成効率が向上することは明らかであるから，制御計算処理部分についても，それを適用することは十分意味のあることである。この部分の定形化はまず処理手続きのモジユール化，標準化，換言すれば処理手続きを出来るだけ標準サブルーチンで置きかえることから始めることになる。DDCおよびSCCを中心とする応用についていえば，論理ブロック図に現われるブロックと対応する程度の大きさで処理機能をまとめて，制御アルゴリズムのサブルーチンパツケージをとり揃えることが適当であろう。これらの標準アルゴリズムを作成したならば，それらを呼ぶための指定，その中で使用するパラメータの指定が骨格となつて，定形的色彩の強いプログラミング言語を構成することができる。

このプログラミング言語では制御アルゴリズムが機能単位となり，その集りが一連の制御計算手続きを指定することになる。制御アルゴリズム番号がラベルの役を果して外からの参照に用いられる。計算に使用される変数の形は一括して宣言しておくべきで，これは初期値の指定，同一記憶域使用の指定などと共に宣言部に書く。標準的なアルゴリズムはシステムに備えつけとして扱い特に定義せずに使い，新しいアルゴリズムを追加する場合の定義とシステムへの登録の方法を与えておく。これによつて，この言語システムの適用面を拡張することが可能となる。

制御計算処理の部分は前述のように多様性に富むために，入出力処理部分程には表現形式の定形化が適さないと考えられる。この部分を

どの程度まで定形化するかは，この言語系の適用対象の広さ，言語の簡易化の要求の強さによつてきめられるが，DDCを中心とする場合でもある程度の適用範囲を保つことを条件とすれば，穴埋め方式に徹するよりはマクロアセンブラあるいはFORTRANの簡単な命令形式の程度に止めるべきである。

3.2.3 言語システムの構成

前述のように制御用プログラムは，プロセス入出力処理と制御計算処理の二部分から形成され，それぞれに適当な定形化言語形をきめ得るが，これを実働させるには，それらの定形化言語プログラムを実行可能な形式に変換するプロセッサと，実行時の制御管理を司るものが必要であり，更にプログラムのデバグおよび修正に用いる補助的な機能プログラムがあることが望ましい。

入出力処理については，実行時にもサブシステムとして動作する半独立的なプログラムが置かれることが適当で，入出力処理指定に書かれた記述を処理するプロセッサは，その独立的入出力プログラムへのパラメータ設定の機能をもつものとなる。

制御計算処理については，ソースステートメントを実行時形式に変換し，それが制御アルゴリズムパッケージを呼びつゝ制御計算を実行する形をとる。この場合，DDCを中心とする応用に於ては，殆ど制御アルゴリズムパッケージの連続で処理が進行する形態となるので，ソースステートメントから作られる実行時形式は，サブルーチン呼出しの情報（サブルーチンの指定とパラメータの集まり）のみから成ることになる。更に，制御用に特有のマルチタスク管理，多様な割込み処理を考慮すると，制御計算のための管理プログラムを置いて，それの管理の下に計算処理の実行時プログラムが進行する形態をとることが適当である。実行時プログラムは，サブルーチン呼出し情報の集まりであるから，これを適当な形式に圧縮しておき，管理プログラムで解読しながら実行を進める解釈方式をとり入れることが，全体の処理系

を簡単にすると考えられる。解釈方式の場合，処理速度の低下の可能性が問題となるが，ここでとり上げている応用面については，実行時プログラムの内容は極めて簡単な情報であることと，サブルーチンは必要に応じて最適化を施して作成できることから，充分実用的なシステムが形成できよう。実用にあたつて制御ループの数と起動周期の兼ね合いが重要となるので，制御プログラム作成者がシステム稼動状況を推定し得るよう，各サブルーチンの大きさ，実行所要時間の情報と制御ループのプログラム実行時間算定法が与えられねばならない。

この様な考え方に基づいて構成したシステムは，表面に現われる言語と共にその実行時プログラム構成と実行する方法に特色があり，具体的なシステムを用いて説明することが全体を明らかにするので，次節にMELCOM−350／30用MDSSについて記述する。

3.3 MDSSの言語とプロセッサ

3.3.1 全体の構成

MDSS（MELCOM DDC & SUPERVISORY CONTROL SYSTEM）は前述の考え方に基づいて構成された制御用の問題向き言語システムである。MDSSは全体として一つのシステムを構成するもので，次のサブ・システムから成つている。

(1) プロセスデータ・モニタ……プロセス入出力処理部

a) アナログ・スキャン・システム

b) デイジタル・スキャン・システム

(2) DDCシステム……制御計算部

これらのサブシステムに対応して，それぞれの言語，言語プロセッサ，実行時プログラム群，およびユーテイリテイ・プログラムがあり，各構成要素相互間の関連を中心として，全体の構成を概念的に示すと図3.2のようになる。

MDSSは一つのトータル・システムであるが，その動作は既存のオ

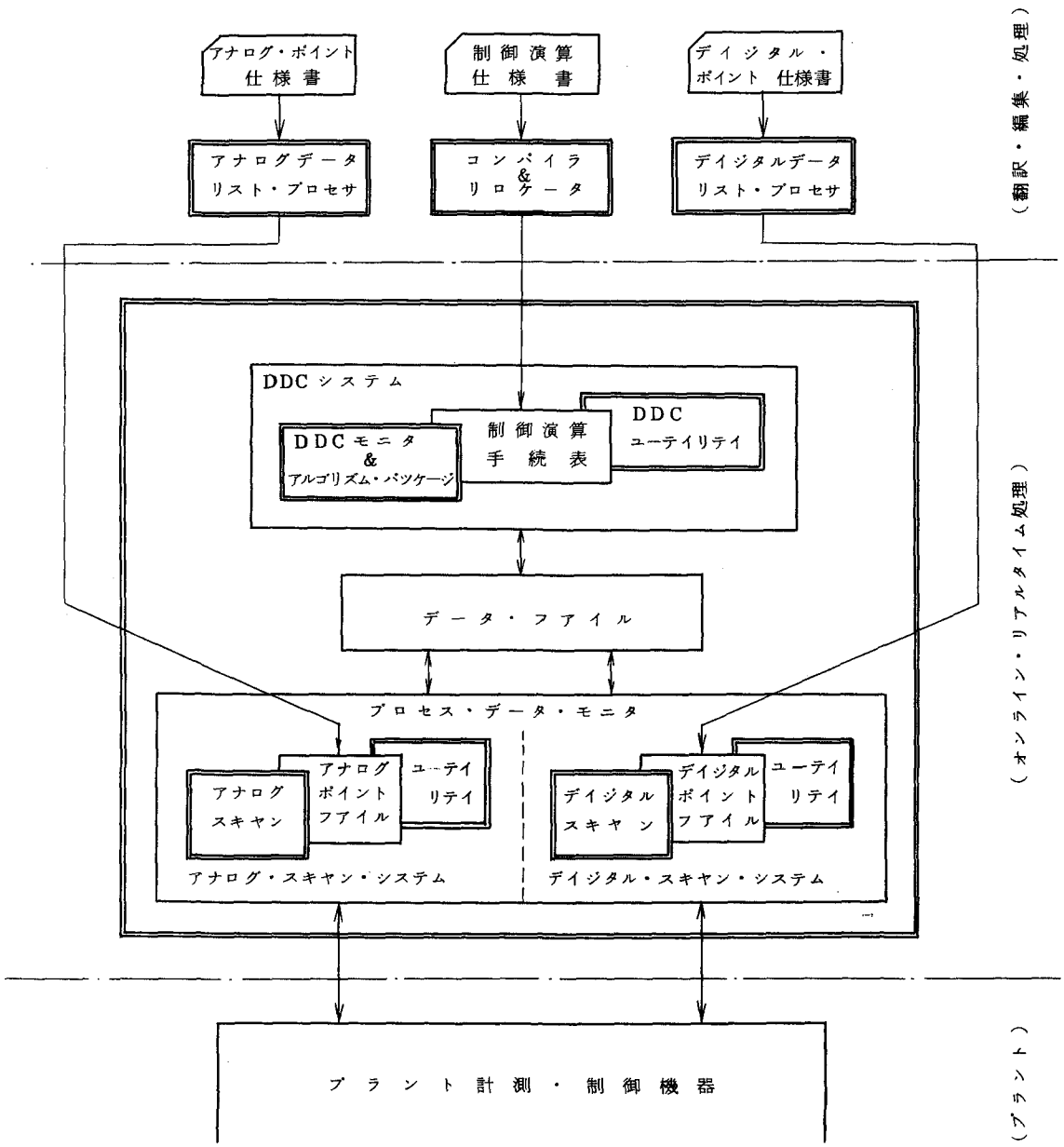


図 3.2　MDSS 構成要素の関連

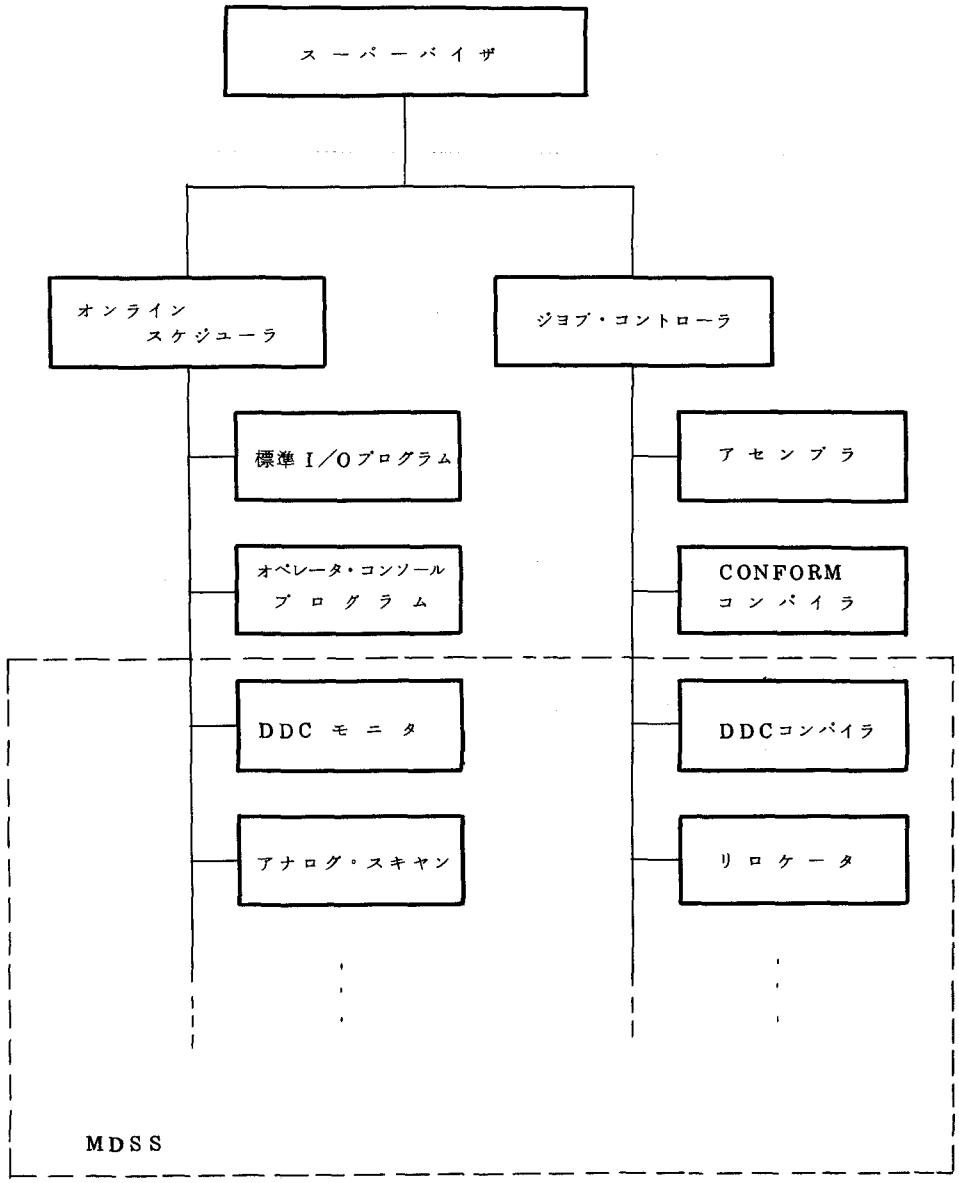


図3.3　MDSSとオペレーティングシステムの関連

ペレーテイング・システムと融合した形で行なわれる。MDSSの構成要素とオペレーテイング・システムの関係は図3.3に示す通りで，実行時系のものがオンライン・スケジューラの系列下に，翻訳時系のものがジョブ・コントローラの系列の下に位置している。MDSSのサブシステムは他の言語（アセンブラ語，CONFORMなど）から作られたオブジエクト・プログラムと共存して動作させることが可能である。従つて，既存のプラント制御用プログラムに部分的にMDSSの要素を組み込んで行くこともできるし，また，MDSSの要素を他の言語のプログラムでつなぎ合わせて，全体のプログラムを構成して使うこともできる。このような配慮は，長期的に集積されたプログラム技術を有効に生かしながら，新しい手法を導入して行く上での必要性に基づいたものである。

3.3.2 アナログ・スキャン・システム

プロセスデータ・モニタとそれに関連する部分のもつ機能は，次の3種に大別できる。

(1) 入出力点ごとにデータの性質と，そのデータに施すべき処理に関する定義をする機能，およびこれらの定義を読み取つて，実行時プログラムに適合する形式に情報を整形する機能，これらはプロセスデータ・リストとそのプロセッサの形で実現される。

(2) 上で与えられたデータの定義に従つてデータの入力，出力を実行し，関連する処理を施す機能。これらはデータをスキャンする実行時のプログラムの形で実現される。

(3) プラント制御のプログラムが準備された以後に，入出力データ点に関する処理の変更などを行なう機能。これらはユーテイリテイ・プログラムとして実現され，オペレータ・コンソールからの介入による修正処理を可能とする。

アナログデータとデイジタルデータについては，実際上の処理が大巾に異なるので，プロセスデータ・モニタをアナログとデイジタル

の2系列に分割した方が良い。アナログ・スキャン・システムは，アナログデータに関してこれらの機能を実現したものである。

アナログデータについては，入力データは数が多く，かつスキャン動作との関連が強いが，出力データは個数が多くなく，また特定の計算処理との対応が強いものが多い。そこでMDSSではアナログデータに関しては，先ず入力スキャンシステムのみを重点的に設け，出力は制御アルゴリズムの1つとして設ける方針をとつた。アナログデータに関する定義および処理指定は図3.4に示す様式の用紙に必要な事項を記入することによつて行なわれる。記入する項目の種類は次の通りである。

a) 入力点シンボルとその説明

b) 入力周期

c) 印字する時の書式

d) スキャナ・リレー・アドレス

e) 工学単位

f) 入力値から工学単位への変換式種類

g) 各種の限界値　上下限値，目標値と偏差，変化率

h) 初期値

i) 代替計器

j) 積算値，平均値などの計算（別様式も併用）

k) その他

アナログ・スキャン・システムの構成要素の関係は，図3.5に示す通りである。前述のアナログスキャンデータリストと共にシステムシンボルリスト（アナログ入力値の格納場所を定義したもの）をアナログデータリスト・プロセッサで処理してアナログポイント・ファイルを作る。これはデータスキャンに必要な情報を網羅したデータブロックの形をもち，SDB(Scan Data Block)と呼ぶ。実行時には，アナログスキャン部にあるプログラム群が，このSDBを参照しながら処理を進める。アナログ入力点に関する各種の情報の追加，変更，削

MDSSアナログスキャン データ リスト

SF：スキャン周期　R：計器レンジインデックス　CT：変換式タイプ
G　：アンプゲインインデックス　A：警報種別　F　：スムージングファクタ
S　：グループスキャン　I：入力番号　UN：プラント番号
C　：チャネル番号　D：警報不感帯インデックス　AIDX：アクションインデックス

作成：＿＿＿＿＿＿　日付：＿＿＿＿＿＿

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80

入力点シンボル SYMBOL	内容説明 DESCRIPTION	工学単位 E-UNIT	同期 SF	印字桁 PRT	リレー番号 RELY	G	S	C	R	A	I	UN	ID	SEQUENCE
													1	0 0 0
													1	2 0 0
													1	4 0 0
													1	6 0 0
													1	8 0 0

入力点シンボル SYMBOL	警報上限値1，基準値 ALM-HILM	警報下限値1，偏差 ALM-LOLM	警報上限値2 P-ALM-HI	警報下限値2 P-ALM-LO	D	CT	F	変化率制限値 RATE-LIM	ID	SEQUENCE
									2	0 5 0
									2	2 5 0
									2	4 5 0
									2	6 5 0
									2	8 5 0

入力点シンボル SYMBOL	計器レンジMIN RANG-MIN	計器レンジMAX RANG-MAX	係数 A_0 COEFF-A_0	係数 A_1(T) COEFF-A_1	係数 A_2(P) COEFF-A_2	係数 A_3(M) COEFF-A_3	ID	SEQUENCE
							3	1 0 0
							3	3 0 0
							3	5 0 0
							3	7 0 0
							3	9 0 0

入力点シンボル SYMBOL	初期値 INIT-VAL	代替計器 SP-INS	AIDX	ID	SEQUENCE
				4	1 5 0
				4	3 5 0
				4	5 5 0
				4	7 5 0
				4	9 5 0

1　11　21　31　41　51　61　71　80

図 3. 4

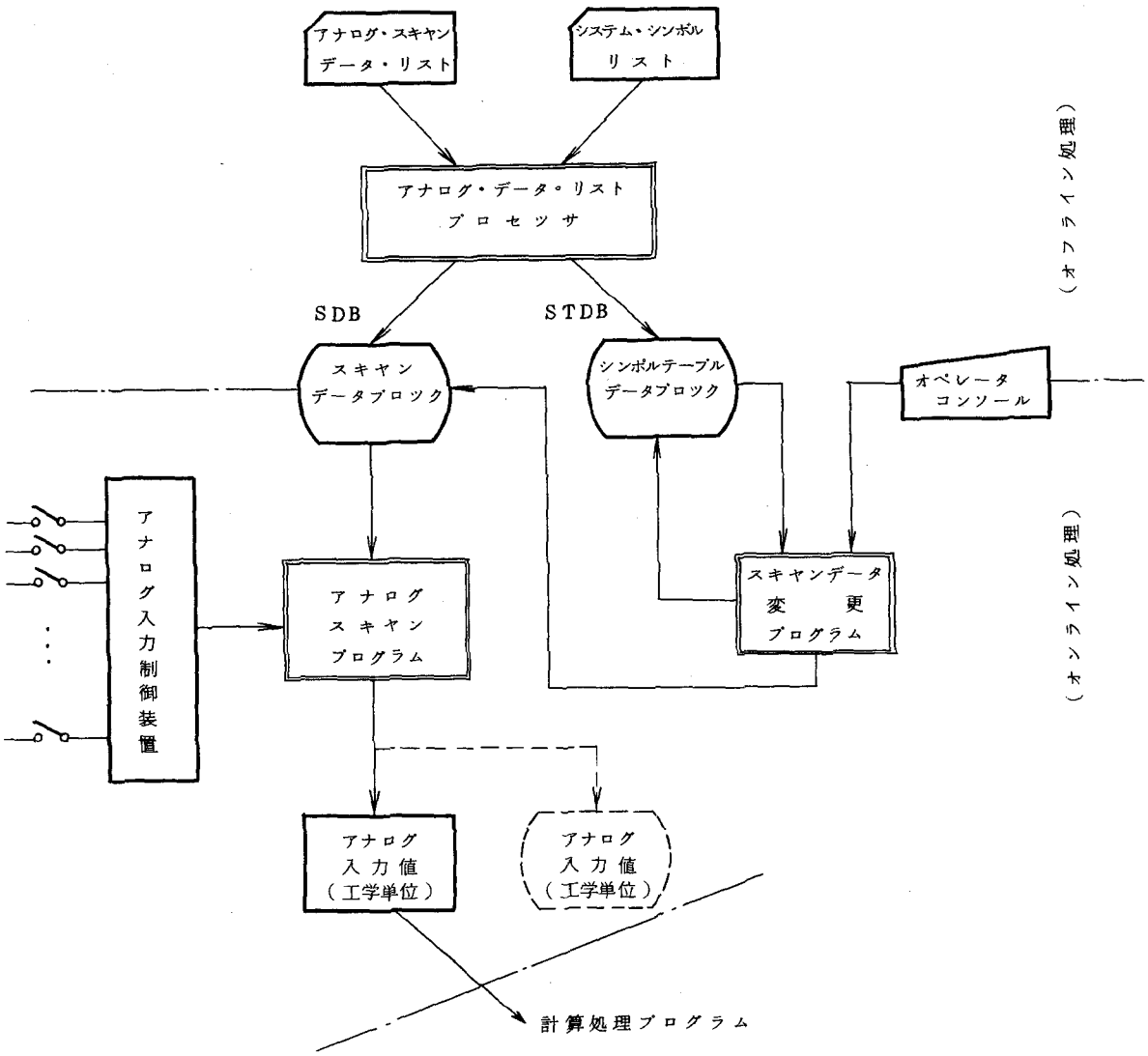


図 3.5　アナログ・スキャン・システムの構成

除はシステムタイプライタからSDBの内容を変更することにより簡単に行なえる。

3.3.3 デイジタル・スキャン・システム

デイジタル・スキャン・システムはアナログ・スキャン・システムと同様の処理をデイジタルデータに対して行なう部分である。MDSSで処理するデイジタルデータはプラントの各部の状態を表わし，あるいは信号を伝えるオン・オフ接点データである。それらのデータに対する処理機能は，接点入力のスキャニング，それに関連したアラームのスキャニング，および接点出力のスキャニングから成つている。

前二者，すなわち接点入力スキャニングとアラーム・スキャニングに関する構成要素の関連を図3.6に示す。接点入力点のデータリストをデイジタルデータリスト・プロセッサで処理して作られた情報がデイジタルポイント・ファイルとして登録される。これは外部メモリに置かれるが，計算機の運転開始時に働く初期化プログラムによつて，主メモリ上の所定の位置に移される。計算機が定常運転に入ると接点入力スキャンプログラムが指定された周期で起動され，接点入力が行なわれる。入力された接点データはイメージテーブル上に置かれる。この過程でアラームチエックの必要を指定されたものについては，データリストで指定された正常値との比較を行ない，不一致があればその接点の番号の情報をアラームメッセージ・バッファ上の行列に登録する。

一方，アラームの処理を司るアラームプリント・プログラムは接点入力スキャンプログラムとは別のタイミングで起動されて，アラームメッセージ・バッファ内の情報を基に外部メモリ上のデイジタルポイント・ファイルからアラーム印字に必要な情報を抽出，成型して所定のタイプライタに印字する。

正常に入力された接点データはイメージ・テーブル上にあり，計算処理の必要に応じてこれが使われる。

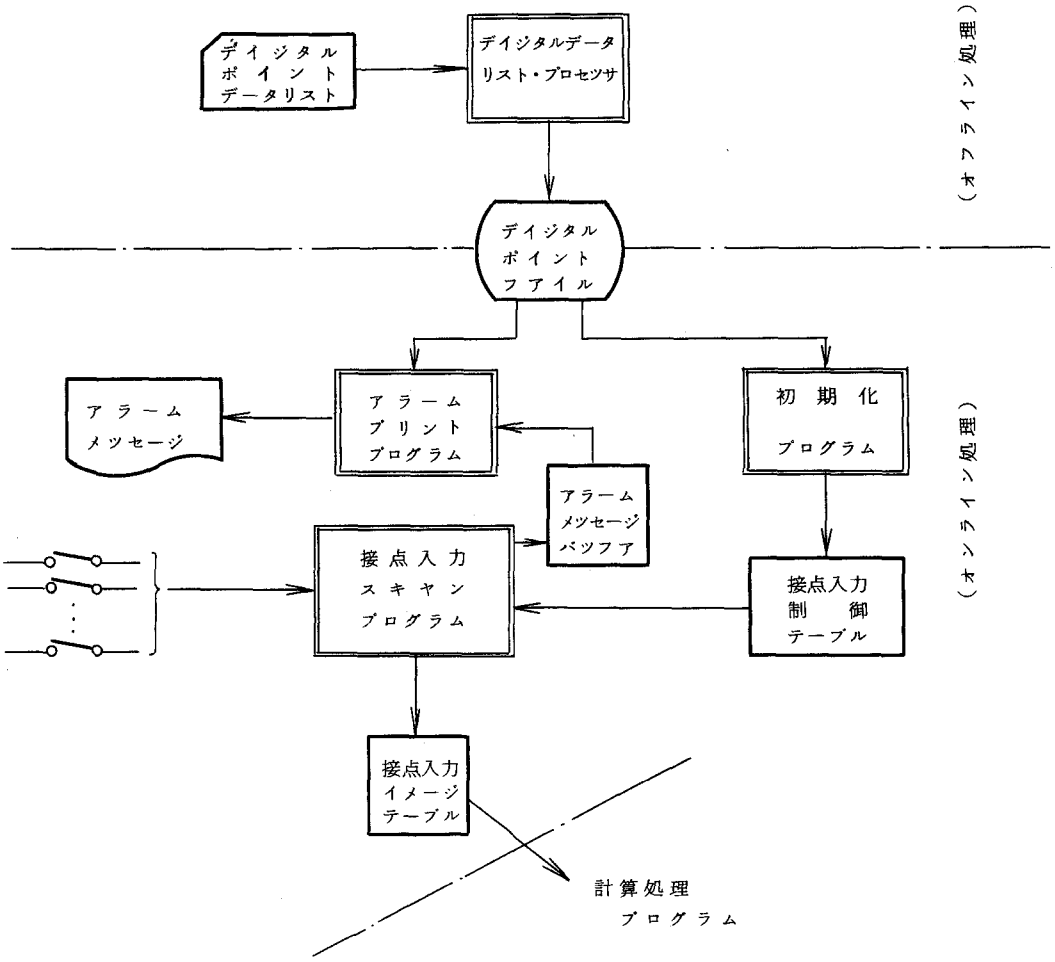


図 3.6　接点入力およびアラームスキャンの構成

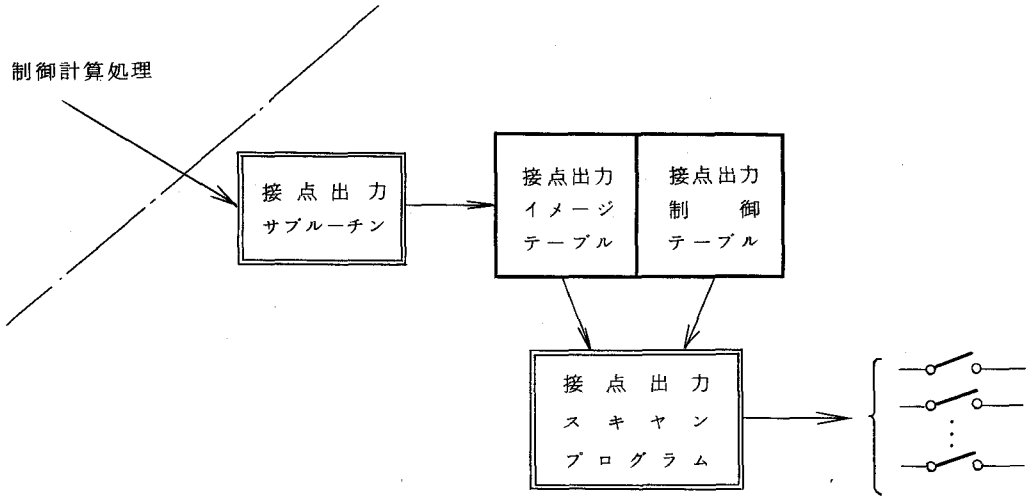


図 3.7　接点出力スキャンの構成

接点出力スキャニングに関する構成要素の関連を図3.7に示す。制御計算処理に伴つて接点出力サブルーチンが働き，接点出力イメージテーブルにデータがセットされると共に，該当する出力グループの出力要求フラグがセットされる。接点出力スキャンプログラムは，別のタイミングで起動されており，出力要求フラグに基づいて出力処理を行なう。接点出力には出力データの変換法（ONに対し〝1〟を出すか〝0〟を出すか，など）により幾通りかの分れがあり，それぞれにサブルーチンが設けられている。

デイジタルデータに関するデータリストは図3.8に示す様式をもつている。記入する処理項目には次の様なものがある。

(1)　点シンボルとその説明

(2)　ビット位置（イメージテーブル上の）

(3)　ステータスの説明

(4)　各種のチエック

MDSS デイジタル・ポイント・データリスト

CCI…SP:設定値(1/0)　　LC: リミットチエック(Y/N)
CCO…S :マニュアルセット(Y/N)　R: マニュアルリセット(Y/N)

作成：＿＿＿＿＿＿　日付：＿＿＿＿＿＿

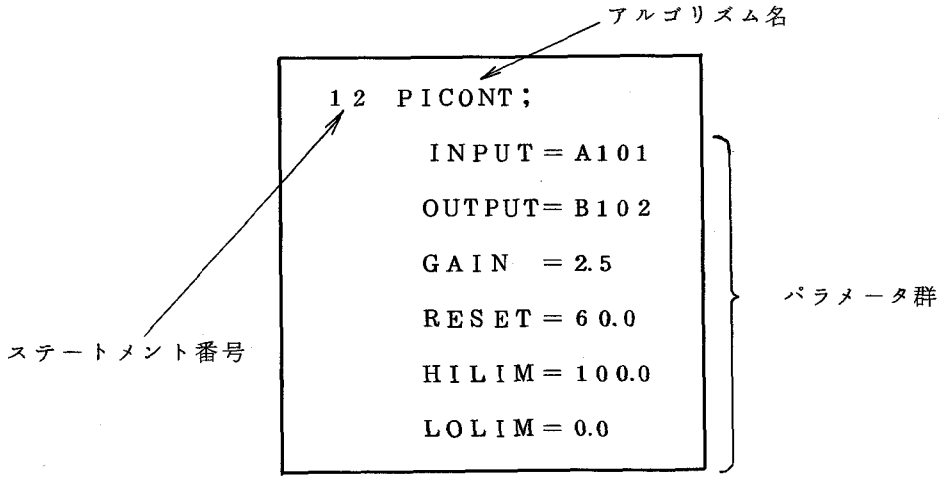
シンボル SYMBOL	内容説明 POINT DESCRIPTION	ポイント番号 BIT REG	ステータスの説明 SET RESET	各種チエック SP/S LC/R	種別 ID	SEQUENCE
1	9	42	46 50 57	65 68	71	73

図 3. 8

3.3.4　DDCシステム（DDC言語）

MDSSの中で制御計算処理に関係する部分をDDCシステムと呼び，そこで用いる言語をDDC言語と呼ぶ。この言語は入出力部分より多様性をもつた処理を記述する必要があるので，前述のプロセスデータ・モニタの様な定形式でなく，やゝ数式表現に近い形式を用いる。

DDC言語の処理記述単位は，MDSSがDDCを中心とした適用を想定していることから，DDCループの記述に用いられるブロック線図中の各機能ブロックに対応したものとし，それをアルゴリズム・ステートメントと呼ぶ。アルゴリズム・ステートメントはステートメント番号，アルゴリズム名，および何個かのパラメータから成る。パラメータの表現には，単にそれらをリストとして並べる方法でなく，その意味を並記した形をとり，プログラムのドキュメンテーション性を高めている。例えばPI制御アルゴリズムは次の様に表現する。



MDSSでは直接デイジタル制御で用いられる典型的な制御アルゴリズムについて，一通りのアルゴリズム・パッケージ・プログラムが用意されており，それらをMDSS標準のアルゴリズム・ステートメントにより使用することができる。これらはFORTRANにおける組込み関数と同じように特に定義することなく使用できる。MDSSの標準アルゴリズムの一覧を表3.1に示す。

表3.1 MDSSアルゴリズム・パッケージ一覧表

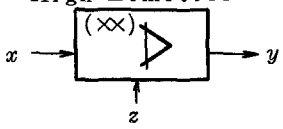
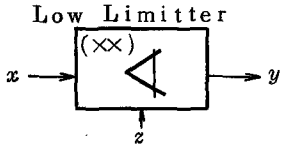
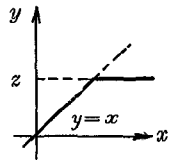
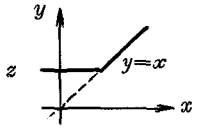
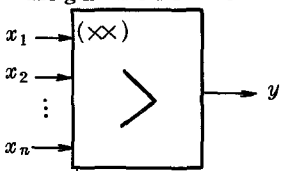
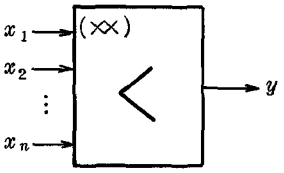
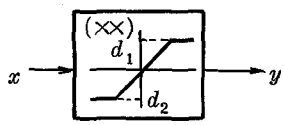
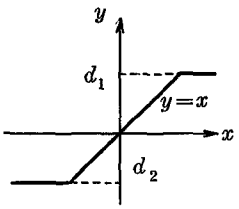
図　記　号	ステートメント	備　考
1. 無条件ブランチ 	×× BRNCH; DEST=	
2. リミッタ High Limitter  Low Limitter 	×× HILIM; INPUT=x OUTPUT=y LIMIT=z ×× LOLIM; INPUT=x OUTPUT=y LIMIT=z	 
3. セレクタ High Selector  Low Selector 	×× HISEL; INPUT=x_1 INPUT=x_2 ⋮ INPUT=x_n OUTPUT=y ×× LOSEL; INPUT=x_1 INPUT=x_2 ⋮ INPUT=x_n OUTPUT=y	$y=\mathrm{Max}(x_1, x_2, \cdots, x_n)$ $y=\mathrm{Min}(x_1, x_2, \cdots, x_n)$
4. クランプ 	×× CLAMP; INPUT=x OUTPUT=y HILIM=d_1 LOLIM=d_2	

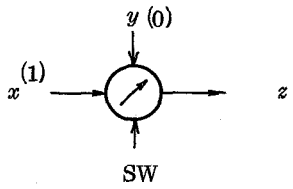
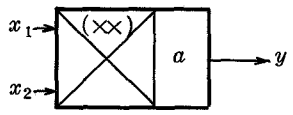
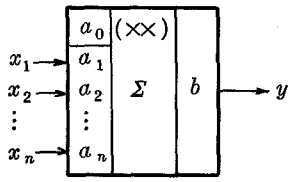
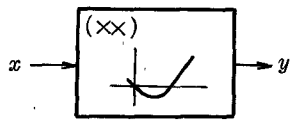
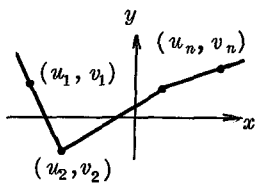
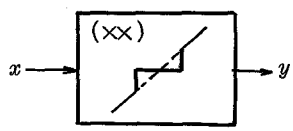
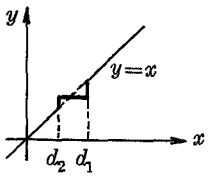
図　記　号	ステートメント	備　　考
5. アナログスイッチ  	××　ANLGSW; INPUT 1 = x INPUT 0 = y OUTPUT = z SWITCH = SW	SW=1 ならば $z=x$ SW=0 ならば $z=y$
6. マルチプライヤ  	××　MULT; INPUT = x_1 INPUT = x_2 OUTPUT = y COEF = a	$y = a \cdot x_1 \cdot x_2$
7. アダー  	××　ADDER; INPUT 1 = x_1 INPUT 2 = x_2 ⋮　⋮ INPUT n = x_n OUTPUT = y COEF 0 = a_0 COEF 1 = a_1 COEF 2 = a_2 ⋮　⋮ COEF n = a_n BIAS = b	$y = a_0 \sum_{i=1}^{n} a_i x_i + b$
8. ファンクションジエネレータ  	××　FUNGEN; INPUT = x OUTPUT = y X = u_1 , Y = v_1 X = u_2 , Y = v_2 ⋮　⋮ X = u_n , Y = v_n	
9. デッドバンド  	××　DEDBND; INPUT = x OUTPUT = y HILIM = d_1 LOLIM = d_2	

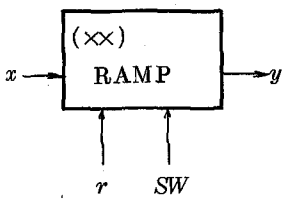
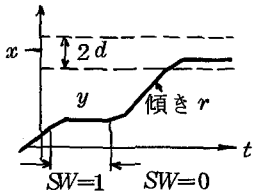
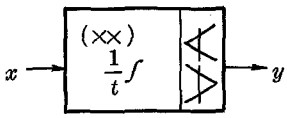
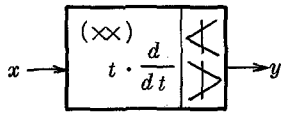
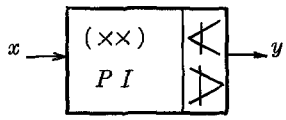
図　記　号	ステートメント	備　　考
10. ラ　グ 	××　LAG； INPUT＝x OUTPUT ＝y RESET＝T	$y(s)=\dfrac{1}{1+TS}\ x(s)$
11. ランプジエネレータ 	××　RAMP； DEMAND ＝x REF　　＝y RATE　＝r DBAND＝d SWITCH ＝SW	
12. 積　分 	××　RESET； INPUT＝x OUTPUT ＝y RTIME＝T HILIM＝d_1 LOLIM＝d_2	$y(s)=\dfrac{1}{T\cdot S}\cdot x(s)$ 但し，$d_2\leq y\leq d_1$
13. 微　分 	××　RATE； INPUT＝x OUTPUT ＝y RTIME＝T HILIM＝d_1 LOLIM＝d_2	$y(s)=\dfrac{T\cdot S}{1+T\cdot S}\cdot x(s)$ 但し，$d_2\leq y\leq d_1$
14. PIコントローラ 	××　PICONT； INPUT＝x OUTPUT ＝y GAIN　＝K_P RESET＝T_I HILIM＝d_1 LOLIM＝d_2	$y(s)=\{K_P+\dfrac{1}{T_I\cdot S}\}\cdot x(s)$ 但し，$d_2\leq y\leq d_1$

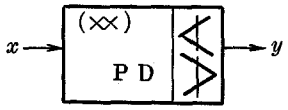
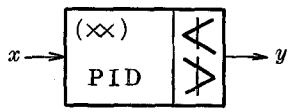
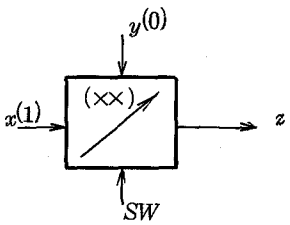
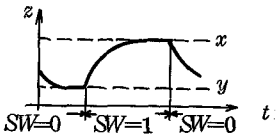
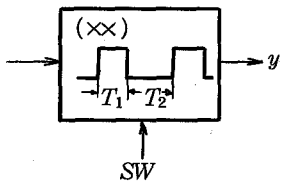
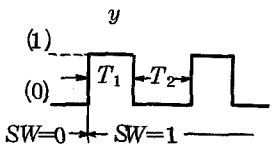
図　記　号	ステートメント	備　　考
15. PDコントローラ 	××　PDCONT； INPUT＝x OUTPUT ＝y GAIN　＝K_P RATE　＝T_d HILIM＝d_1 LOLIM＝d_2	$y(s)=\left\{K_P+\dfrac{T_d\cdot S}{1+T_d\cdot S}\right\}\cdot x(s)$ 但し，$d_2\leq y\leq d_1$
16. PIDコントローラ 	××　PIDCNT； INPUT＝x OUTPUT ＝y GAIN　＝K_P RESET＝T_I RATE　＝T_d HILIM＝d_1 LOLIM＝d_2	$y(s)=\left\{K_P+\dfrac{1}{T_I\cdot S}+\dfrac{T_d\cdot S}{1+T_d\cdot S}\right\}\cdot x(s)$ 但し，$d_2\leq y\leq d_1$
17. アナログゲートスイッチ 	××　ANTRNS； INPUT1＝x INPUT0＝y OUTPUT＝z RESET　＝T SWITCH＝SW	一次遅れ付きアナログスイッチ 
18. パルスジエネレータ 	××　PLSGEN； OUTPUT＝y TIM　1　＝T_1 TIM　2　＝T_2 SWITCH＝SW	

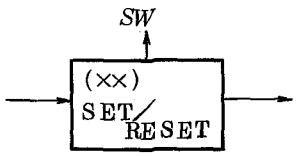
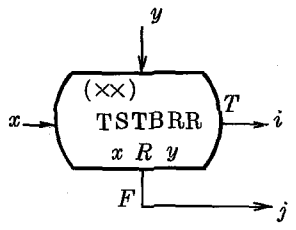
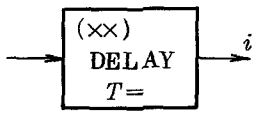
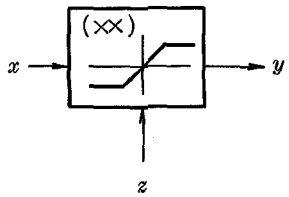
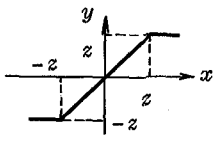
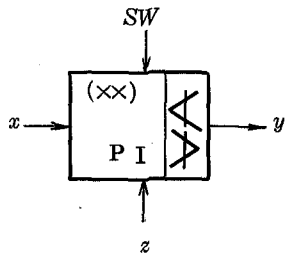
図　記　号	ステートメント	備　　考
19. セット／リセットロジカル 	××　SETRST； SET　　$=SW_1$ SET　　$=SW_2$ ⋮ RESET $=SW_1'$ RESET $=SW_2'$ ⋮	論理変数の値を セット(1)又はリセット(0)する
20. テストブランチ 	××　TSTBRR； INPUT $=x$ REF　　$=y$ COMP　$=R$ TRUE　$=i$ FALSE $=j$	x と y を R で比較 {真ならば i へ {偽ならば j へ R：EQ，GT，LT etc.
21. ディレイ 	××　DELAY； DTIME $=T$ RETURN $=i$	指定時間，プログラムの実行を停止。
22. 対称クランプ  	××　SYMCLMP； INPUT $=x$ OUTPUT $=y$ LIMIT $=z$	
23. A／M 変換付きPIコントローラ 	××　PIAM； INPUT $=x$ OUTPUT $=y$ GAIN　$=K_P$ RESET $=T_I$ HILIM $=d_1$ LOLIM $=d_2$ TRACK $=z$ AMSW　$=SW$	(1) A／M SW=MANU 初期値を z でトラッキング (2) A／M SW=AUTO 通常のPI制御

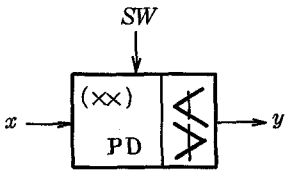
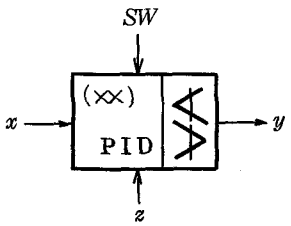
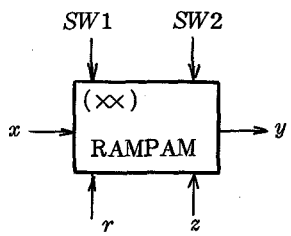
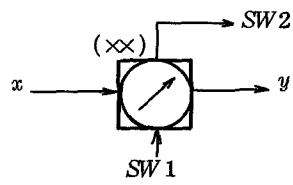
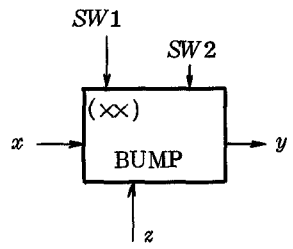
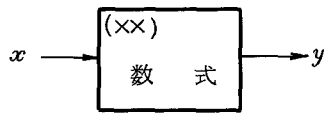
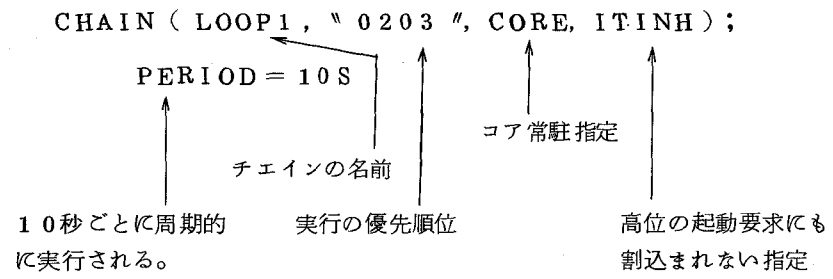
図　記　号	ステートメント	備　　考
24. A/M 変換付きPDコントローラ  	××　PDAM； INPUT＝x OUTPUT＝y GAIN　＝K_P RATE　＝T_D HILIM＝d_1 LOLIM＝d_2 AMSW　＝SW	(1) A/M SW=MANU 初期値セット (2) A/M SW=AUTO 通常のPD制御
25. A/M 変換付きPIDコントローラ  	××　PIDAM； INPUT＝x OUTPUT＝y GAIN　＝K_P RESET＝T_I RATE　＝T_D HILIM＝d_1 LOLIM＝d_2 TRACK＝z AMSW　＝SW	(1) A/M SW=MANU 初期値をzでトラッキング (2) A/M SW=AUTO 通常のPID制御
26. A/M 変換付きランプジエネレータ  	××　RAMPAM； DEMAND＝x REF　＝y RATE　＝r DBAND＝d HOLDSW＝$SW1$ TRACK＝z TRKSW＝$SW2$	(1) トラッキングモード ($SW2$=0) yの初期値をzでトラッキングする (2) コントロールモード ($SW2$=1) ランプ信号を発生
27. A/M 変換スイッチ  	××　TRNS AM； INPUT＝x OUTPUT＝y SWITCH＝$SW1$ SETBIT＝$SW2$	MANUモードにおける設定値出力をしや断するためのアルゴリズム。 (1) MANUモード($SW1$=0) 0→$SW2$ (設定値出力なし) (2) AUTOモード($SW1$=1) { 1→$SW2$ x→y (設定値出力あり)

図　記　号	ステートメント	備　　考
28. レイト付きバンプレス トランスファ  	××　BUMP； MV　　＝z SP　　＝x ADJUST＝y RESET＝T LBAND＝d AUTO　＝$SW1$ HOLD　＝$SW2$	(1) MANUモード($SW1$=0) (設定値x－制御値z) をトラッキング，この時 $y=z$ (2) AUTOモード($SW1$=1) yの値をリセットタイ ムTで設定値xに近づ ける (3) $\lvert x-y \rvert \leq d$ になれ ば一次遅れに切りかえ る。
29. 演　算  	××　COMPUTE； $y_1=f(x_1)$ $y_2=f(x_2)$ ⋮ $y_n=f(x_n)$	四則 論理 } 演算が可能 (例) A＝B＊(C－D)／E L＝SW.OR. (K.AND.P)

ブロック線図上の各ブロックの接続状況に対応して，パラメータを選びながらアルゴリズム・ステートメントを並べると，それらのつながりで構成される一つの処理単位が出来る。これをチエインと呼び，これがDDCシステムにおける実行上の取扱い単位となる。オペレーティング・システムとの関連でいえば，このチエインがタスクに相当する。

チエインの先頭には，その実行の条件を規定するチエイン・ステートメントを置く。これは例えば次の様なものである。



DDC言語のプログラムは

MACRO DEFINITION

PROGRAM DECLARATION

CONTROL CALCLILATION

の３部分から構成する。このうちMACRO DEFINITION はオプションで，システムに新しいアルゴリズム・ステートメントを定義する場合に書かれる。ここでは新しいアルゴリズム・ステートメントの形式に関する定義をするのみであり，対応する実行時サブルーチンは別に用意せねばならない。

PROGRAM DECLARATION 部では，次に示す宣言ステートメントを用いて，プログラム名，変数名のタイプ，初期値などを指定する。

宣言ステートメント	パラメータ	意　　味
PROGRAM;	名前	プログラム名
REAL;	変数名	実数型変数名
INTEGER;	〃	整数型変数名
LOGICAL;	〃	論理型（ワード）変数名
BIT;	〃	論理型（ビット）変数名
EQUIVALENCE;	変数名＝変数名	変数名の再定義
INITIAL;	変数名＝変数名	変数名の初期値指定

CONTROL CALCULATION 部は実際の制御計算処理を記述する部分で，前述のチエインを複数個列記することができる。

DDC 言語によるプログラムの形式は図3.9に示す通りである。

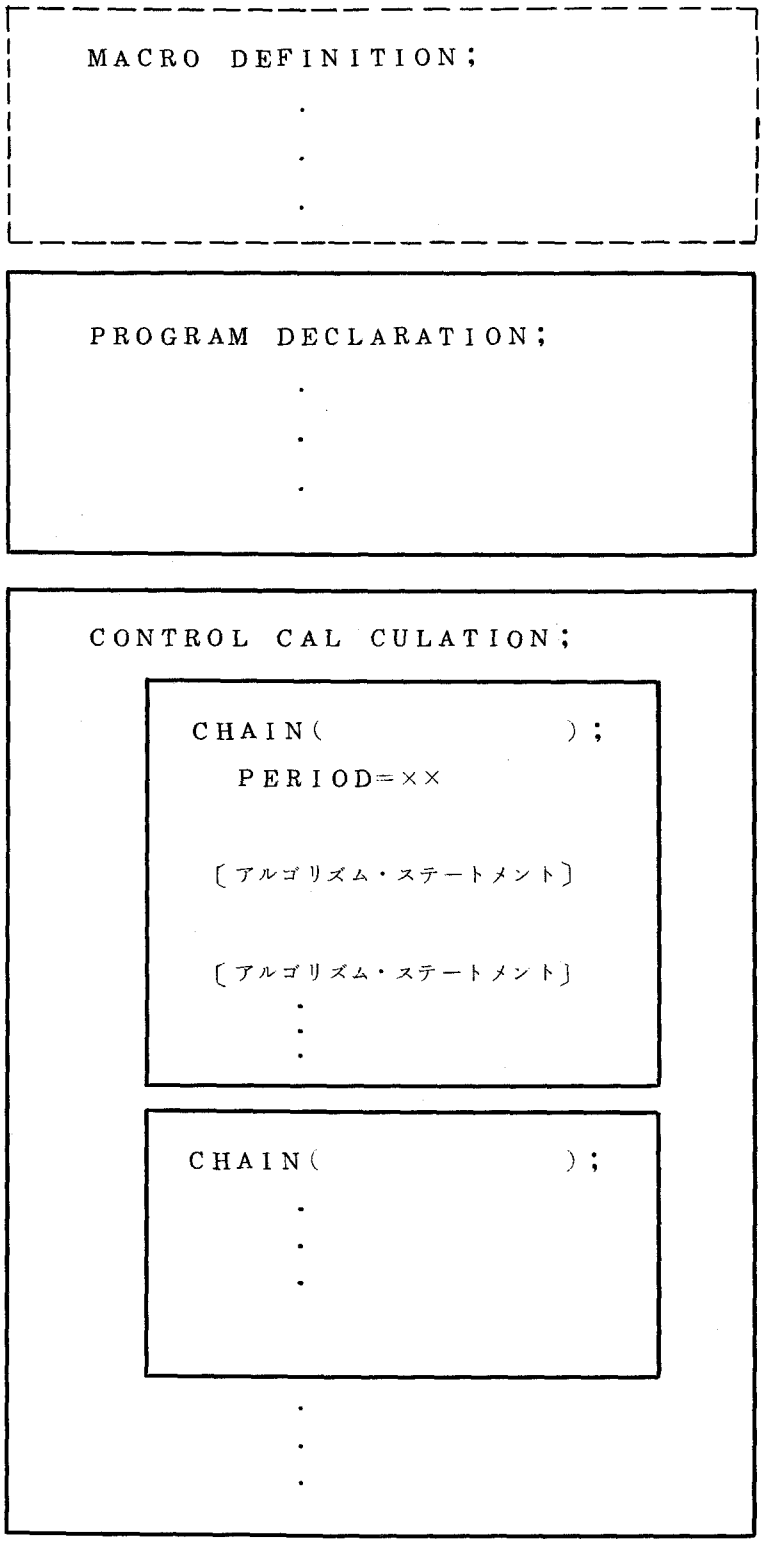


図 3.9　DDC言語プログラムの形式

DDC 言語のシンタックスを拡張Backus Normal Formで表現したものを本章付録に掲げておく。実際にこの言語を適用した例として，ブロック線図を図3.10に，それに対応するプログラムを図3.11にそれぞれ示す。

上の例はマクロ定義を含んでいないが，これを含む例として図3.12のブロック線図でFUNCTION GENERATORをマクロとして定義する場合のプログラムを図3.13に示す。

3.3.5 DDCシステム（オブジェクト・プログラム）

上述のDDC言語で書かれたソース・プログラムはトランスレータによつて実行時形に変換される。このオブジェクト・プログラムは，制御アルゴリズム・パッケージを呼び出す情報と，その中で使うパラメータの集まりから成る。その形式と構成は，DDCプログラムの実行制御方式との関連できめられる。

MDSSではアルゴリズム・パッケージを呼ぶ情報とパラメータをテーブル形式にまとめ，デイレクタと呼ばれる解釈ルーチンにより，その解読と実行を制御する方式をとつている。これは，オブジエクト・プログラムとしてサブルーチン呼出し命令を直接並べる方法をとると，実行速度上は有利であるがプログラムの保守性は悪くなり，かといつて個々のサブルーチン呼出しをタスクとして分離すると，タスク数の増加によるタスク管理のオーバーヘッドの増加が重大になる事情を考慮した策である。解釈方式による速度低下は，各制御アルゴリズムの実行が充分最適化を施したサブルーチン内で行なわれ，解釈ルーチンではテーブルで与えられた情報の簡単な処理により，該当サブルーチンを選択するだけを行なうので余り問題とならない。

オブジエクト・プログラムおよびそれが参照するデータ領域のメモリ内の位置を固定せず，再配置の融通性を残すことが好ましいので，オブジエクト・プログラムにはリロケータブル形式とアブソリュート形式の2通りが設けられている。両者とも構造上はMELCOM−350／

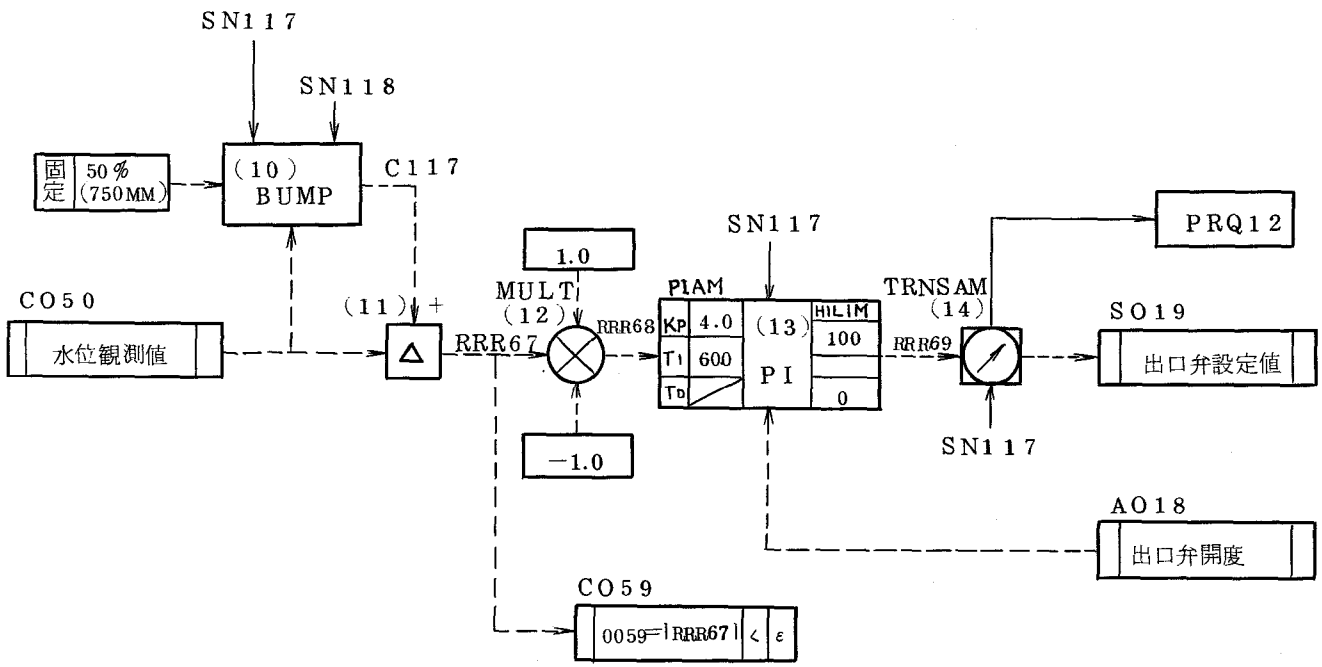


図 3.1 0　典型的な定値制御（水位制御）ブロック図

```
PROGRAM DECLARATION;
  PROGRAM; WTRSPR      :WATER SEPARATOR CONTROL:
  REAL ;   CO50        :WATER SEPARATOR LEVEL
           CO59        :CONTROL ERROR
           SO19        :WATER DRAIN VALVE SET POINT
           AO18        :WATER DRAIN VALVE OPENNING
  BIT;     SN117       :WD VALVE CONTROL A/M SWITCH
           SN118       :BUMP GO/HOLD SWITCH
CONTROL CALCULATION;
  CHAIN(WDVALV,"020A", DRUM, ITINH);
       DELTATEE=2S
  10 BUMP;
       MV       = CO50
       SP       = 50.0
       ADJUST   = C117
       RESET    = 100.0
       LBAND    = 2.0
       AUTO     = SN117
       HOLD     = SN118
  11 COMPUTE;
       RRR67    = C117-CO50
       CO59     = ABS(RRR67)
  12 MULT;
       INPUT    = 1.0,  = RRR67
       OUTPUT   = RRR68
       COEF     = -1.0
  13 PIAM;
       INPUT    = RRR68
       OUTPUT   = RRR69
       GAIN     = 4.0
       RESET    = 60.0
       HILIM    = 100.0
       LOLIM    = 0.0
       TRACK    = AO18
       AMSW     = SN117
  14 TRNSAM;
       INPUT    = RRR69
       OUTPUT   = SO19
       SWITCH   = SN117
       SETBIT   = PRQ12
  15 EXIT;
END;
```

図 3.11　コーディング例

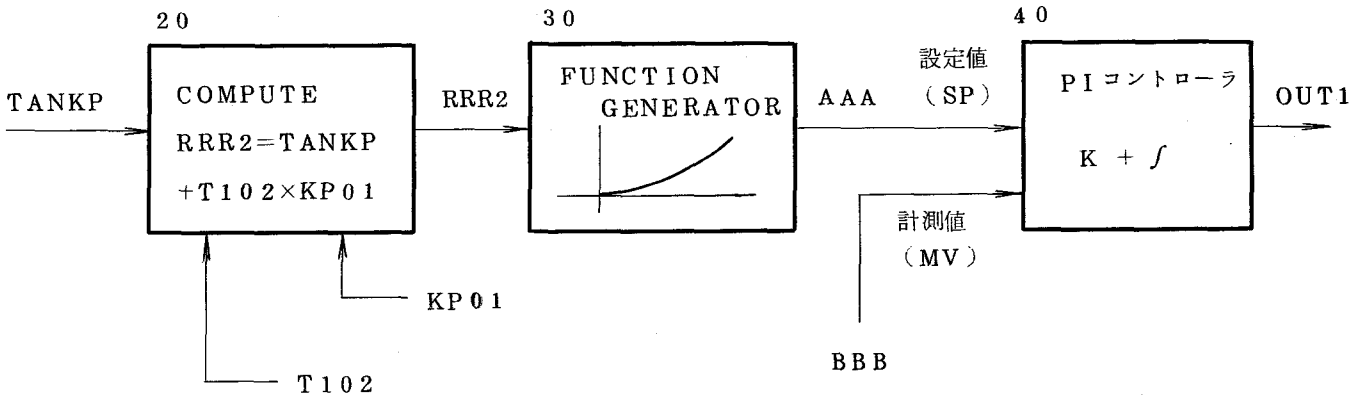


図 3.12　FUNCTION GENERATORを含む制御ループ

```
MACRO DEFINITION;
    FUNCGEN(ALGNO = 12, NCORE = 20);
        INPUT(FORM = V, TYPE = R)
        OUTPUT(FORM = V, TYPE = R)
        X(NPAR = -10, FORM = C, TYPE = R)

PROGRAM DECLARATION;

    PROGRAM;   SAMPLE      :PROGRAM NAME :

    REAL;      TANKP       :TANK PRESSURE :
               T102        :COEFF. 1      :
               KP01        :COEFF. 2      :
               AAA         :SET POINT     :
               BBB         :MEASURED VALUE:
               OUT1        :VALVE POSITION:

    INITIAL;   TANKP = 120.0
               T102 = 10.0
               KP01 =  2.5

CONTROL CALCULATION

   CHAIN(LOOP03,"051A",  DRUM);
        PERIOD = 1M

      20   COMPUTE;
                 RRR2 = TANKP+KP01×T102

      30   FUNCGEN;
                 INPUT = RRR2, OUTPUT = AAA
                 X = 0.0,          Y = 0.0
                 X = 10.0,         Y = 2.0
                 X = 85.0,         Y = 80.0
                 X = 100.0,        Y = 100.0

      40   PICONABS;
                 MV = BBB
                 SP = AAA
                 GAIN = 1.2
                 RESET = 0.2
                 OUTPUT = OUT1
                 HILIM = 100.0
                 LOLIM = 0.0

      50 EXIT;

END;
```

図 3.13 プログラム例

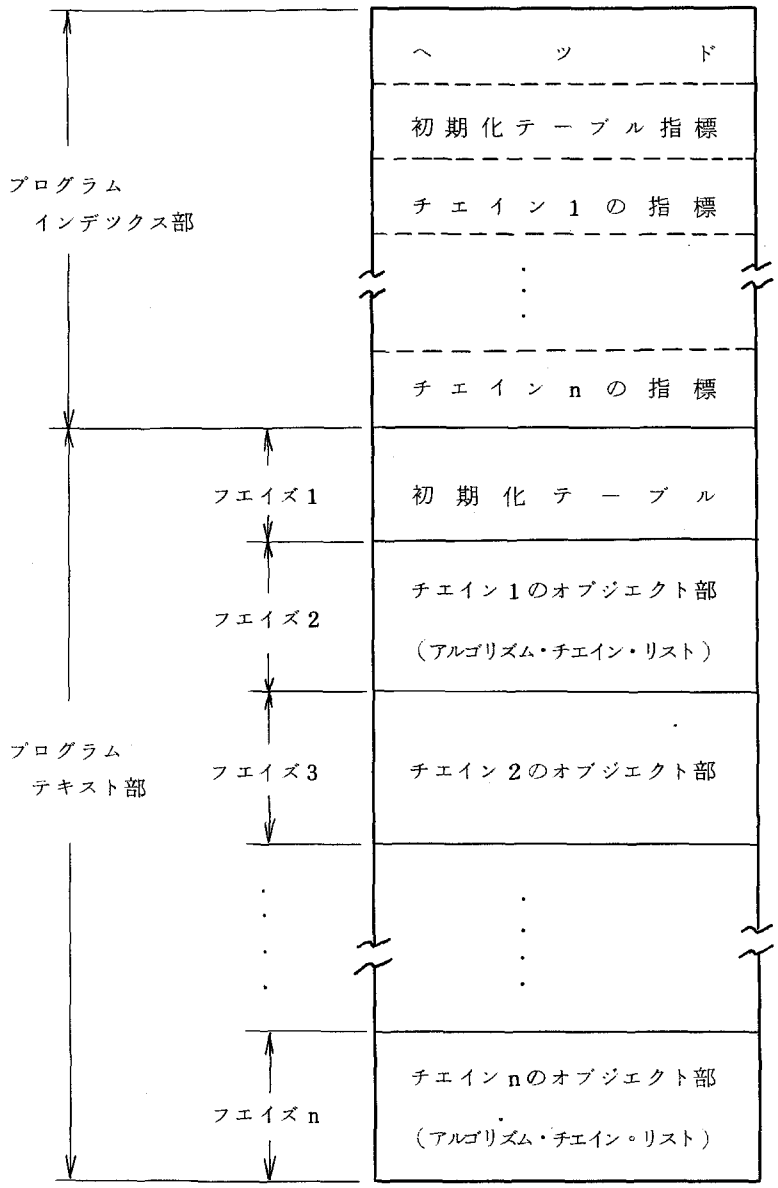


図3.14　アブソリュート・オブジェクト・プログラムの構造

30のオペレーテイング・システムにおけるロード・モジユールと同じになつており，他の言語によるプログラム部分と共通の管理を受け，また既存の各種ユーテイリテイ・プログラムの利用を可能としている。

DDCシステムのアブソリユート・オブジエクト・プログラムの構造は図3.14に示す通りである。リロケータブル・オブジエクト・プログラムはこれと類似の形をもち，シンボル・テーブルおよびリロケーションのための情報を含んでいることと，各情報の収容位置が若干ちがう点だけこの形と異つている。

プログラム実行時に実際にメモリ内に入つて動作するのは，プログラム・テキスト部のフエイズ2以降であり，各フエイズには個々のチエインに対応するアルゴリズム・チエイン・リストが入つている。これは図3.15に示すアルゴリズム・ステートメントのオブジエクト形の集まりである。図3.15のオブジエクト形でアルゴリズム番号は対応する制御アルゴリズム・パッケージの位置を示す情報であり，これをもとにコントロールがパッケージの方へ移される。パラメータはパッケージで必要とする変数のアドレスまたは定数である。

ステートメント番号	アルゴリズム番号
パラメータ・ワード1	
パラメータ・ワード2	
・ ・ ・	
パラメータ・ワードn	

図3.15 アルゴリズム・ステートメントのオブジエクト形式

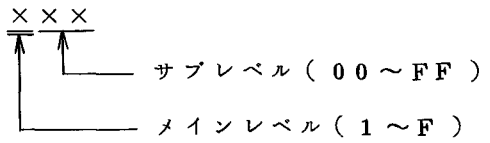
図3.14のフエイズ1にある初期化テーブルは各チエインのレベル番号，起動周期，変数の初期値から成り，プログラム登録あるいはシステム初期化の際にこれらの情報が使われる。プログラム・インデツクス部は，プログラム・テキスト部の各部分のローデイングに必要な情報を集めた部分で，各部分の名前，在り場所，サイズなどから成る。この部分はオペレーテイング・システムに於けるロード・モジユールの形式を踏襲するために形を合わせてあり，ヘツドという。このDDCシステムでは，必要

なものもダミーとして挿入してある。

3.3.6 DDCシステムの実行

DDC言語のプログラムはチエインの集まりで構成される。各チエインはDDCにおける個々の制御ループに対応しており，実用上，制御ループ毎に動作の状態（作動の周期，起動のタイミングなど）が異なるので，DDCシステムのプログラム実行の制御管理はチエイン単位で行なわれる。

各チエインには演算処理のタイミング上の重要度に応じて，優先順位が割付けられる。プログラムを構成するチエインの個数は多数に及び，オペレーテイング・システムが持つ優先順位数より多くなり得るため，ここでは優先順位系を拡大して，メインレベル最大15，サブレベル最大256（各メインレベルについて）とする。プログラム記述のときには，レベルの指定を3桁の16進数で表わす。これがレベル番号で，実行時にはこれによつてチエインの識別を行なう。



チエインの実行は後述のDDCモニタによつて管理されるが，チエイン間の優先順位制御は次のスケジユール規則に従う。

(1) メインレベル・スケジユール

あるレベルのチエインを実行中により，高レベルのチエインに対する実行要求が起きた場合，実行中のチエインは中断され，高レベルのチエインが起動される。

中断されたチエインは，実行要求が出ているより高いレベルのすべてのチエイン（中断中に発生したものを含め）を実行した後，中断点から再開される。

以上のチエイン割込みは，アルゴリズム・ステートメント実行の切れ目で行なわれる。

(2) サブレベル・スケジユール

同一メインレベル内のサブレベル間では実行中の割込みはなく，高位のサブレベルのチエインから順次（実行要求の出ているものについて）実行される。但し，メインレベルの割込みによつて中断していたチエインがあればそれがそのメインレベル内で最優先となる。

チエインの実行の制御管理と，DDC システム以外のサブシステムとの実行時のインターフエイスを司ることのためには，幾つかのプログラムおよびテーブルを用意する必要がある。これらのものを集めて制御管理用のモニターブログラムを構成し，これを DDC モニタと呼ぶ。DDC モニタは図3.16に示す構成をもち，スーパーバイザの下に位置して働く。各々の要素の概略内容および機能は次の通りある。

(1) ステータス・テーブル

各チエインの動作状態を示す情報をもつコアメモリ上のテーブルで，実行中のチエインだけでなく実行し得る全てのチエインについて用意され，次の内容をもつ。

a) 起動要求の有無

b) delay 中か否か

c) 実行可能か否か

d) 割込み禁止モードか否か

e) チエインのスタートアドレス

(2) タイム・テーブル（オプション）

一定時間間隔で起動されるべきチエイン，または delay 中のチエインに対し，必要な時間データを持つコア上のテーブル。

(3) グループリクエスト・テーブル（オプション）

あるイベントの生起に同期して起動されるべき，チエイン群のレベル番号をもつコア上のテーブル。

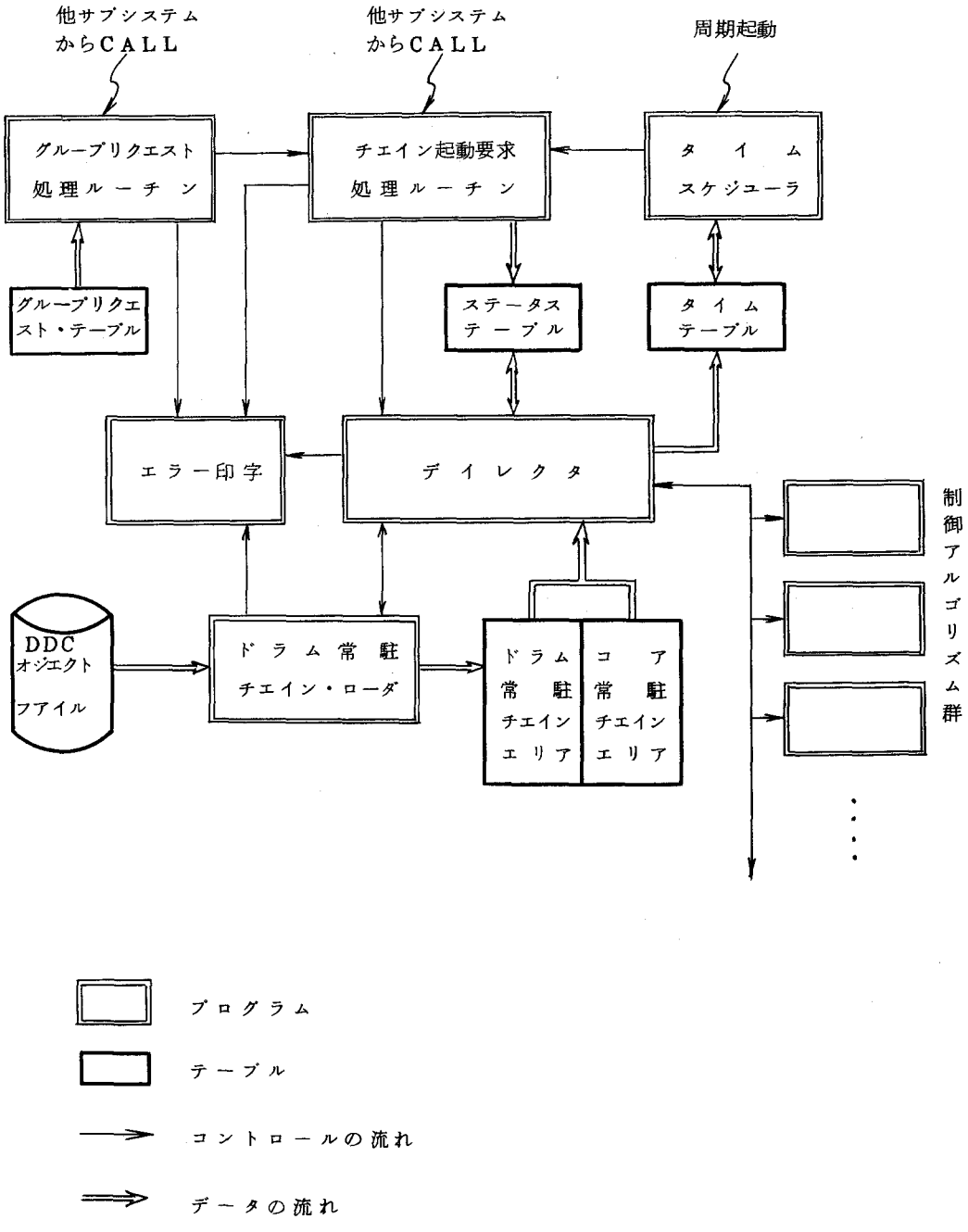


図 3.16　DDC モニタの構成

(4) ディレクタ

チエインの実行を直接制御するプログラム，ステータス・テーブルを調べる起動要求が出ているチエインについて，アルゴリズムステートメントを解読し，実行のコントロールを該当するアルゴリズムルーチンへ移す。一つのアルゴリズムの実行が終るとコントロールはディレクタに戻り，ステータステーブルを調べて，より高位のチエインの起動要求があればそちらの実行に移る。

アルゴリズムを次々実行し，一つのチエインの実行が終り，他に起動要求が全くない場合は終止する。

ディレクタの起動は通常次にあげるチエイン起動要求処理ルーチンにより行なわれる。

(5) チエイン起動要求処理ルーチン

チエインの起動要求を受付けるルーチンで，ステータステーブル中の該当するビットを立て，ディレクタが動作中でなければこれを起動する。

(6) グループリクエスト処理ルーチン（オプション）

チエイングループに対する起動要求を処理する。ルーチンで，チエイン起動要求処理ルーチンを使つてステータステーブルをセットする。

(7) タイムスケジューラ（オプション）

一定時間間隔（例えば1秒毎）で起動され，タイムテーブル中の時刻の更新を行なう。起動予定時刻に達したチエインがあれば，チエイン起動要求処理ルーチンを使つて起動要求を出す。

3.3.7 ユーテイリテイ・プログラム

プロセス制御のユーザ・プログラムは一般に多くの部分から成つており，プラントの現地据付けと並行して計算機メモリへの組込みが行なわれる。プラントが稼動に至るまでには，部分的な試験を積み重ねて行く関係上，これらのプログラムを一時的に変形したり，他のプロ

グラムモジュールとの接続関係を変則的にしたりする処置が必要となる。プログラムのパラメータの中には，据付調整の段階ではじめて固定できるものもある。さらに運用に入つて後にも，一部の装置の故障や運転法の変更に伴つて応急的にプログラム・パラメータを変更する必要に迫られることがある。このようにプラント制御のプログラムはある程度の手直しが避けられない実情にある。

この問題は，プログラムの修正という消極的な見方をするより，オンラインシステムに特有のシステムのチューニングとして積極的に捉えて，このためのソフトウエア・ツールを用意することを考えるべきであろう。

実際の処理はプログラムの修正変更となるが，この場合，現地設置計算機がプログラムの再コンパイルに必要な機器構成を持たず，最小限の補助機器しか使えないことが重要な問題である。

プロセス制御応用におけるプログラム・チューニングに要求される機能は次の諸項である。

(1) ソース言語レベルでの処置が可能であること。これは高位言語を用いる場合に重要で，プログラムの内部構造あるいは計算機ハードウエアの知識を前提とせずに，プログラムパラメータのチエツクや変更をするための条件である。

(2) プログラムの一時的な実行抑制，あるいは周期的な実行など，プログラム実行に関する制御の機能があること。

(3) オンライン制御と並行してチューニング操作が行なえることが望ましい。

(4) 現地計算機の構成のままで処理可能なこと。

(5) チューニング操作の記録を正確に残せること。

(6) 誤操作の可能性が少ないこと。

MDSSでは上述の機能をユーテイリテイ・プログラムを組込む形で実現している。これはアナログ・スキヤン・システム，デイジタル・

スキャン・システムおよびDDCシステムの各部分にそれぞれ付加されているが，何れも最も基本的な入出力装置であるシステムタイプライタのみを用いて処理を可能とし，また会話モードにより，処理を進める方式を採用して人と機械のコミュニケーションを便利な形にすると共に履歴の記録性を良くしている。

アナログおよびデイジタルのプロセス変数のモニタリング処理システムに関するユーテイリテイ・プログラムの主要機能は，次の通りである。

(1) スキャニングの停止，再開

(2) モニタリングの停止，再開

(3) 警報上下限値，変化率限界値の変更

(4) 初期値の変更

(5) ポイント情報の表示および手動設定

DDCシステムに対するユーテイリテイ・プログラムの主要な機能は次の通りである。

(1) チエインの単発的あるいは，周期的な起動要求とその停止

(2) チエインの実行禁止とその解除

(3) チエインへの割込み禁止とその解除

(4) チエインの消去

(5) チエインの内容変更と内容表示

(6) 変数の初期値設定

DDCシステムのユーテイリテイ・プログラムの使用例として，アルゴリズム・パラメータの変更を行なつたときの操作と記録を図3.17に示す。図3.17(a)のプログラム中のPICONTアルゴリズムのパラメータGAINの値を2.5から1.0に変更するときの手続きが図3.17(b)である。下線を施した部分は計算機からの出力である。

///SYMDはこの機能ルーチンの呼出しであり，MDSSFファイル

```
PROGRAM DECLARATION;
  PROGRAM; FELCNT :FUEL CONTROL:
  REAL; CV101  :CALCULATED VALUE:
        SP101  :SET POINT:
        AI101  :MEASURED VALUE:
        VA5    :CONTROL VALUE:
               .
               .
               .
CONTROL CALCULATION;
  CHAIN(LOOP1, " 0301 ", DRUM);
               .
               .
               .
  40 PICONT;
       INPUT = RRR1,     OUTPUT = VA5
       GAIN = 2.5,       RESET = 300.0
       HILIM = 100.0,    LOLIM = 0.0
               .
               .
               .
```

図 3.17 (a)　プログラム例

```
:MP:///SYMD, MDSSF;FELCNT
CHAIN = LOOP1, D
ST.NO.= 40, PICONT
ARG   = GAIN
DATA  = 1.0
PLOC  = 28B5(35A8;4)
OLD   = 41280000
NEW   = 41100000
ARG   = :CEND
```

図 3.17 (b)　パラメータ変更の例

にあるプログラム名FELCNTなるプログラムが対象であることを指定する。次に，チエイン名がLOOP1であること，ドラム上にあるプログラムを対象とすることを指定する。次にアルゴリズムステートメント番号が40，その名がPICONTであることを指定する。更に変更するパラメータGAINを呼出し，値として1.0を入れると，そのアドレス（ドラム上の），前の値および新しい値をそれぞれ16進数で打ち出して来る。

この様な記録はシステムタイプライタ上の紙に残るが，その前にはオペレーテイング・システムの働きによつて，作業実施日時などデータロギングに必要な情報が出されているから，プログラム変更やそれに関連する各部状態のプリントがシステムのチユーニングの記録として利用できる。

3.4 結　　言

プロセス制御用の問題向き言語システムの構成について，言語と実行システムの両面から述べた。

このシステムの実用上の最も顕著な効用は，プログラム・コーデイング労力の軽減であり，ボイラ起動シーケンスと温度制御に適用した例では，制御ロジック図を与えられる条件で，12個の制御ループに対するプログラム約1.5 kWを3名が1日以内に完成している。このように計算機およびプログラミング言語の詳細な知識を要せず，応用技術者に親しみのある表現で直接にプログラム作成できるシステムを備えることにより，コーデイングに食われる時間と労力をシステム綜合設計など質の高い作業に振り向け得ること，また簡明な記法でプログラムを管理することにより，不必要な誤りの発生を防ぐことが期待される。

MDSSのような問題向き言語システムは，FORTRANのような手続き向き言語と比べると，実行時系をどの様に組み立てるかという問題が大きな重みを持ち，言語の構成よりオペレーテイングシステムの構成として捉えるべき点が多い。DDCモニタの設計如何によりシステムの応答性の限

界，タイム・オリエンテッドかイベント・オリエンテッドかの方式上の適性に差が出てくる。実行時システムとしてはなるべく広範囲の適用を可能にしたいが現実には充分な性能を発揮し得る範囲は限られているので，適用対象によつて幾つかの実行時システムを備え，アルゴリズム・パッケージなどを共用する方針が必要であろう。

第３章　付　録

MELCOM−350−30/30F MDSS DDC言語のシンタツクス

1．Backus Normal Form の拡張

BNFによる表現を簡略化するため，次の記法を導入する。

(1)　・・・

・・・の左に書かれた一つの要素を連続して並べることを示す。

(2)　〔　〕

〔　〕でかこまれた要素が一つの要素として扱われ，かつ〔　〕内の要素全体が省略可能であることを示す。

(3)　{　}

{　}でかこまれた一連の要素の並びが一つの要素として扱われることを示し，同時に{　}内が空でない意味ももたせる。

2．MDSS　DDC言語のシンタツクス

2.1　基本要素

letter ::=A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|
Q|R|S|T|U|V|W|X|Y|Z

digit ::=0|1|2|3|4|5|6|7|8|9

hexadecimal-digit ::=digit|A|B|C|D|E|F

bit ::=0|1

character ::=letter|digit|△|=|+|−|*|/|,|;|:|$
|"|.|(|)

separator ::=,|(CR)

（注）　△は空白を，(CR)は改行を示す。

2.2 要　　素

integer ::= digit・・・

hexadecimal-constant ::= \ hexadecimal-digit・・・"

bit-constant ::= B \ bit "| .TRUE.| .FALSE.

character-constant ::= C \ character・・・ "

time-constant ::= {〔integer H〕〔integer M〕〔integer S〕
〔integer C〕}

integer-constant ::= 〔+|−〕integer

unsigned-fixed-constand ::= integer.|〔integer〕.integer

unsigned-floating-constant ::= unsigned-fixed-constant
E integer

fixed-constant ::= 〔+|−〕unsigned-fixed-constant

floating-constant ::=〔+|−〕unsigned-floating-constant

logical-constant ::= bit-constant|hexadecimal-constant
|integer

unsigned-real-constant ::= unsigned-fixed-constant|
unsigned-floating-constant

real-constant ::= fixed-constant|floating-constant

constant ::= integer-constaut|hexadecimal-constant|
bit-constant|real-constant|character-
constant|time-constant

identifier ::= letter〔{letter|digit}・・・〕

temporary-variable ::={RRR|III|LLL}{letter|digit}・・・

simple-process-variable ::= identifier

process-variable ::= simple-process-variable〔(integer)〕

variable ::= temporary-variable|process-variable

comment ::= :character・・・{:| (CR) }

2.3 プログラム

```
program ::= {[macro-definition][data-declaration
            control-calculation]}END;
```

2.3.1 マクロ定義部

```
macro-definition
  ::= MACRO DEFINITION;algorithm-statement-
      definition[algorithm-statement-definition・・・]
algorithm-statement-definition
  ::= identifier(algorithm-attribute-list);
      [parameter-definition-list][corstant-
      calculation-list]
algorithm-attribute-list
  ::= ALGNO = integer[,algorithm-attribute]・・・
algorithm-attribute
  ::= {NCORE|NCONST|EXTIM}=integer|RETURN=
      {SYS|ARG}
parameter-definition-list
  ::= parameter-definition[separator parameter-
      definition]・・・
parameter-definition
  ::= identifier({[form][type][repeat][calculation
      -table]})
form ::= FORM={C|V|CV|VC}
type ::= TYPE={R|I|L|B|S|C[(integer-constant)]
      |T [({STD|MIN|SEC})]}
repeat ::=NPAR=integer-constant
calculation-table ::=CALT=$TP$(integer)
constaut-calcvlation-list
```

::= calculation-expression[separator calculation-expression]・・・

calculation-expression

::= calculation-variable={calculation-primary {+|−|*|/|**}calculation-primary |[+|−] calculation-primary | calculation-function}

calculation-variable ::=$ {TP|RC}$[(integer)]

calculation-primary

::= calculation-variable|unsigned-real-constant|DT

calculation-function

::= {SIN|COS|ALOG|ATAN|SQRT|EXP|ABS} (calculation-primary)

2.3.2 プログラム宣言部

data-declaration

::=PROGRAM DECLARATION; PROGRAM; identifier [variable-type-declaration][equivalence] [initial]

variable-type-declaration

::= {REAL|INTEGER|LOGICAL|BIT};identifier [separator identifier]・・・

equivalence

::=EQUIVALENCE;equivalence-pair[separator equivalence-pair]・・・

equivalence-pair

::= process-variable=(simple-process-variable [,simple-process-variable]・・・)

initial ::=INITIAL; v-c-pair [searator v-c-pair]・・・

```
v-c-pair
  ::= process-variable[(integer)]=initial-constant
initial-constant
  ::= real-constant|integer-constant|hexadecimal-
      constant|bit-constant
```

2.3.3 制御演算部

```
control-calculation ::=CONTROL CALCULATION;
      chain・・・
chain::=chain-statement algorithm-statement・・・
chain-statement
  ::= CHAIN([chain-name,] level-number [,chain-
      attribute]・・・);[execution-timing-list]
chain-name ::= identifier
level-number ::= hexadecimal-constant
chain-attvibute ::= CORE|DRUM|ITINH
execution-timing-list
  ::= timing [separator timing]・・・
timing ::= {PERIOD|DELTATEE}=time-constant
      |TRIGGER={bit-process-variable|integer}
algorithm-statement
  ::= macro-definition-algorithm|special-algorithm
macro-definition-algorithm
  ::= statemeut-number algorithm-name
      [(INITIAL=initial-constant [, initial-
      constant]・・・)];[algorithm-parametcr[separator
      algorithm-parameter]・・・]
statement-number ::= integer
algorithm-parameter
```

::= parameter-name={parameter-data [separator=
parameter-data]･･･|*}

parameter-data

::=constant|variable|statement-number

initial-constant

::= real-constant|integer-constant|hexadecimal-
constant|bit-constant

special-algorithm

::= statement-number CONPUTE; assignment
[separator assignment]･･･

assignment

::= real-assignment|integer-assignment|logical-
assignment

real-assigument

::=real-variable={real-expression|integer-
expression}

integer-assignment

::= integer-variable={real-expression|integer-
expression}

logical-assignment

::= logical-variable=logical-expression

real-expression

::= [+|-]real-term[{+|-}real-term]･･･

real-term ::= real-factor[{*|/}real-factor]･･･

real-factor

::= real-primary[** {real-primary|integer-
primary}]･･･

real-primary

::= unsigned-real-constant|real-variable|(real-

```
        expression)|real-function({real-expression|
        integer-expression})
real-function
  ::= SIN|COS|SQRT|ATAN|ALOG|EXP|ABS|FLOAT
integer-expression
  ::= [+|-]integer-term[{+|-}integer-term]...
integer-term
  ::= integer-factor[{*|/}integer-factor]...
integer-factor
  ::= integer-primary[**integer-primary]...
integer-primary
  ::= integer|integer-variable|(integer- expression)
      integer-funetion({real-expression|integer-
      expression})
integer-funetion ::=IABS|IFIX
logical-expression
  ::= logical-term[{.OR.|.EOR.}logical-term]...
logical-term
  ::= logical-factor[.AND. logical-factor]...
logical-factor ::=[.NOT.]logical-primary
logical-primary
  ::= logical-constant|logical-variable|(logical-
      expression)
```

第4章　オンライン計算機におけるプログラムシミュレーション(26)～(29)

4.1　緒　　言

本章ではプログラムの検査の効率改善の問題に触れる。これは計算機言語処理システムに補助的な意味で関連するもので，応用プログラムの作成過程では重要な事項である。

制御用計算機ではプログラムの現地調整が不可欠である。その場合，プログラムの変更，デバッギングのための諸操作に際して動作上の制約が厳しいのが普通であり，一部の装置が未稼動の状態でそれらを使用するプログラムを等価試験すること，稼動中のオンライン制御に外乱を与えることなくプログラムテストを空いた装置を用いて行うことなどが要求される。このためにプログラムを安全にシミュレートするシステムが要求される。

ここではシミュレーションの対象となる機能とそのシミュレーションの方法，実用上の効果および問題点をMELCOM－350／30　のために開発されたオンラインシミュレータを実例として述べる。

4.2　デバッギングエイドとしてのシミュレーションシステムの機能

4.2.1　デバッギングの方式

マルチプログラミング計算機で実行されるプログラムは，コアメモリ中で割付けられる位置がロードされる度ごとに変動すると考えなければならない。そのためにプログラムのデバッギングの方法にもそれに対する処置を要する。

この方法の一つとして，被テストプログラムあるいは関連する管理プログラムにデバッギングのための情報，たとえばコアメモリダンプの領域のシンボリック指定などを盛り込むやり方がある。これによりテスト時にオペレータの詳細な手だすけなしに，あらかじめ指定された情報に従つて計算機からデバッギングに必要な情報を出力することができる。この方法は，情報の盛込み方によつて

(1) ジョブコントロール・プログラムに与える。

(2) プログラムリンケージの段階でデバッギング用の管理プログラムを結合させる

(3) あらかじめ被テストプログラムの中にデバッギング用の補助命令を組込んでおく

などに分れるが，いずれも他のシステムプログラムの助けを借りるか，デバッギング完了後にプログラムを本番用に作りかえることが必要である。

デバッギングのための上記と異る方式として，シミュレーションプログラムを別に用意して，これに被テストプログラムの実行を受け持たせ，デバッギングの情報はこのシミュレータを介して与える方法が考えられる。これによればデバッギングのために被テストプログラムに特別な手当てを施す必要もなく，またシミュレータに充分なデバッギング機能を賦与しておくことにより，あたかも計算機を独占しているかのようにプログラムデバッギングを進めることが可能となる。

また，制御用計算機では計算機を現地設置する以前からプログラムテストを進めておくことが要求され，それには計算センタにある機械を使用せねばならず，その機器構成，メモリ内の領域割付けが現地設置予定のものと異なる場合が多い。このため被テストプログラムで使用される入出力装置，メモリ領域を適宜空いているところに置きかえて実行させる必要がある。これに対してはシミュレーションによる方法が効果的であり，被テストプログラムに手を加えることなくこの要求をみたすことができる。

4.2.2 オンラインシミュレータに要求される基本機能

オンライン制御計算機用のデバッギングエイドは，オンラインで稼動中の制御動作と併行してデバッギングを進めること，入出力装置およびメモリ領域の代替使用を行なうことが基本的に要求されるために，動作形態がシミュレーションというべきものになる。通常プログラム

のシミュレーションというと，ある計算機のプログラムを別の計算機で働かせるための道具あるいは方法を指すが，ここでは同一機種のプログラムを環境を変えて働かせることを指している。

この種のシミュレータプログラムでは，環境の制約に関係をもたない命令は何ら手を加えずに直接ハードウエアで実行させ，制約にふれるものだけを選別して必要な変換あるいは代替えの動作をさせればよく，別機種のプログラムのシミュレートの場合に比して速度の低下をおさえることが可能である。さらに，制約にふれる命令の選別，その後の処置のために役立つ基本機能をあらかじめ機械の設計段階からハードウエアおよび管理プログラムに組込んでおくと，シミュレーション機能および性能が飛躍的に向上する。しかしここでは機械および管理プログラムが標準のものとして与えられた場合について論ずる。ハードウエアまであらかじめ手を加える場合は，単にデバッギングだけでなく，診断機能など一連の信頼度向上の手段も併せて考慮されるべきであろう。

オンラインシミュレータの設計上主眼となる事項は次の諸点である。

(1) シミュレータに対する入力データにどのような誤りがあつても，シミュレーション実行時にプログラムの暴走を起こさず，オンライン制御系を乱さない。

(2) 使用者には被テストプログラムを直接実行する感じを与えるよう工夫する。このために，定式的な操作はシミュレータに自動的に処理させる。

(3) デバッギングのための機能を充実する。

(4) 適用システムの多様性を考慮し，融通性をもたせる。

(5) シミュレータ自体をできるだけコンパクトにまとめる。

4.2.3 シミュレーション方式

シミュレーションにも種々の方式があり，計算機のハードウエアおよび管理プログラムその他のソフトウエアの能力に応じて幾つかの特

長ある方式が提案されている。例えば管理プログラムにシミュレーションの制御に便利な機能を付加し，被テストプログラムをその他のプログラムと区別して扱うことを基本とするやり方とか，被テストプログラムをマクロアセンブラを利用して等価な別プログラムに変換するやり方などが見られるが，これらの方式は前述のように個々の計算機あるいは基本ソフトウエアの特別な機能を前提としており，標準的に提供されるシステムの範囲をこえている。

もう一の方法としてインタープリータ方式がある。これは被テストプログラムを1ステップずつ取り出して，解読しながら等価な動作をプログラムで実現するもので，融通性に富んでおり任意のステップでデバッギングのための手続きを挿入することができ，また被テストプログラムの誤りを実行前に検出してプログラムの暴走をくい止めるには最適である。反面実行時間が長くかゝることが欠点である。しかし，ここで問題にしているシミュレータではプログラムを同一機種でシミュレートするものであるから，被テストプログラムの各ステップの点検には時間を要するがその実行は多くの場合直接ハードウエアにより行うことができるから，この欠点をかなり低減できるものとなる。さらに，このシミュレート処理を優先度の高いプログラムの実行と併行して実施すれば，機械の使用効率低下を気にせずにプログラムテストを進めることが可能である。

4.3 MELCOM−350における実用例

上述の考え方に基いて制御用計算機MELCOM−350／30 で開発されたオンラインシミュレータについて，その機能と特に問題となる点を述べる。

4.3.1 オペレーティングシステムとシミュレータとの関係

シミュレータは，オペレーティングシステムの中では図4.1に示すようにコンパイラ，アセンブラと同列に位置し，オンラインプログラムの空時間を縫つて実行される。図から明らかなように，シミュレー

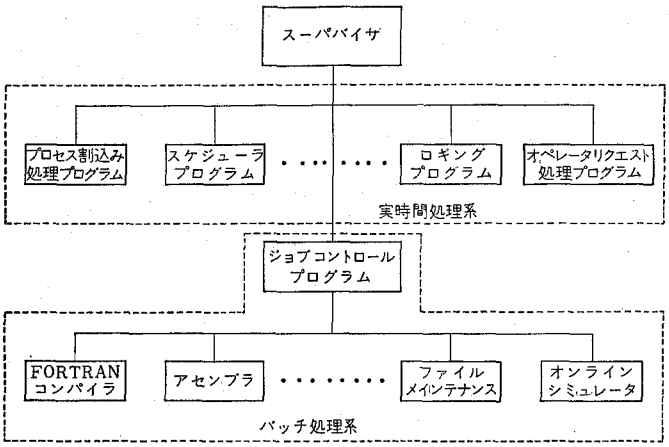


図 4.1　MELCOM−350/30 オペレーテイングシステムの構成

タはスーパーバイザの特別な援助を必要としないので計算機の設置先条件を問わず，どのシステムにも簡単に組込める。

4.3.2　シミュレータの動作モード

シミュレータは図 4.2 に示すように 6 個の状態があり，オペレータモードを中心にして動作する。すなわち，オペレータモードでシミュレータに対し制御情報を入力すると，その内容に従つて各状態に移行し，指定された動作を終了するか，あるいは途中で誤り処置などのためにオペレータの介入を必要とする場合にはオペレータにもどる。

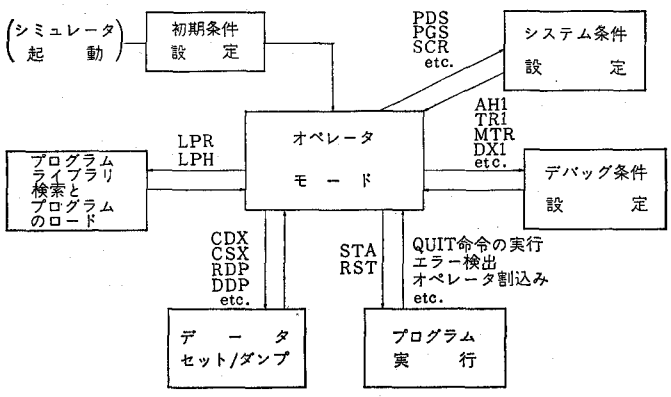


図 4.2　シミュレータの動作状態

4.3.3 シミュレータの取扱う対象プログラム

シミュレータで取扱うプログラムは，

(1) 非特権モードのプログラム

(2) シングルタスクのプログラム

に限定する。特権モードのプログラムを対象に含めるとステータスの変化，メモリの保護キイなどを扱わねばならず，これらをインタープリータ方式で実現するのは負担が大きすぎ，シミュレータが大きくなつてオンラインプログラムとの同時実行におけるコアメモリの占有，退避（ロールイン／アウト）に要する時間の増大など好ましくない問題を生じる。特権モードのプログラムはスーパーバイザのように特殊なものであるのが普通で，ユーザプログラムを主対象とするこのシミュレータの処理範囲外と考える。

マルチタスクに対しては上記と同様，シミュレータの大きさの点で問題がある。ユーザプログラムデバッグの目的ではマルチタスクの動作を直接実現しなくても，タスクごとに分離してそれぞれをシングルタスクとしてシミュレートすることでほとんど充分である。

以上のような制限をつけるが，このシミュレータではスーパーバイザ機能の主要なもの，入出力動作，共通情報のアクセスなどに対するシミュレーション機能を備えているので効果的なシミュレーションが可能である。

4.3.4 CPUのシミュレーション

シミュレータは，プログラムの実行に基本的に必要な次のCPU機能をシミュレートする。

命令の実行

レジスタ

メモリ保護

自動アドレス変換

タイマ機構

各機能のシミュレーションの方法を次に述べる。

(1) 命令の実行

インタープリータ方式を採用しているので，大部分の命令のシミュレーションは次のステップをとる。

a) 不当命令のチエック

b) インデックス，間接番地修飾

c) オペランド・アドレスのチエック

d) 命令を機械に実行させる

e) 必要ならば結果を擬似レジスタに残す

表 4.1 シミュレータの対象とする機械命令

Type / Operation	RR	RS	RI	RX				
LOAD	LR	L	LI	LL	LC	LD	LF	LE
ADD	AR	A	AI	AL	AC	AD	AF	AE
SUB	SR	S	SI	SL	SC	SD	SF	SE
AND	NR	N	NI	NL	NC		SBM	
OR	OR	O	OI	OL	OC		RBM	
BIT OPERATION	*1		TMI					
COMPARE	CR	C	CI	CL	CC	CD	CF	CE
BRANCH	*2		*6				BBS	
STORE REGISTER	BALR	SX	BAL	SXL	SXC		BBR	
LOAD REGISTER	LXR	LX	LXI	LXL	LXC			
ADD REGISTER	AXR	AX	AXI	AXL	AXC			
MULTIPLY	*3	M	MI MLI	ML	MC	MLL	MF	ME
STORE	*4	ST		STL	STC	STD	STF	STE
DIVIDE	*5	D	DI	DL			DF	DE
			RDD					

*1 BIT OP. (CB, RB, TB, ICB, CLR, IB, CSB, SB)
*2 SKIP OP. (SCR, SCS)
*3 RIGHT SHIFT OP. (SRL, SRR, SRA, DRL, DRR, DRA)
*4 REGISTER OP. (CML, CMA, SVC, STS, ATX)
*5 LEFT SHIFT OP. (SLL, SNL, SLA, SNA, DLL, DNL, DLA, DNA)
*6 BRANCH OP. (BCR, BCS)

命令の動作そのものを機械に直接実行させている点がこの方式の特徴である。スーパーバイザ呼出しなどの特別な命令についてはそれぞれ専用ルーチンを設けて等価な動作を実現させる。

このシミュレータが対象としている命令の種類を表 4.1 に示す。この他の命令として，特権命令と可変長データ命令があるが，前者は前述の理由で，後者は制御用としての使用頻度が小さい理由で，対象外としてシミュレータを単純化している。

(2) レジスタ

インタープリータ方式では1命令を実行ごとにシミュレータが介入するので，命令の演算結果を保持するために擬似レジスタが必要となる。シミュレータは通常のプログラムが使用する全レジスタに対して擬似レジスタを設けており，演算結果はすべてこの擬似レジスタに残される。

(3) メモリ保護

ハードウエアが備えているメモリ保護機能は異常検出後の処置がスーパーバイザに任されてしまうためシミュレータには不向きである。シミュレータのメモリ保護機能は実行前に命令のオペランドアドレス（インデックス，間接番地修飾後）が自己に与えられた領域または共通情報領域以外であればその旨を表示してオペレータの指示を待ち，常にシミュレータの世界からの逸脱を防いでいる。

(4) 自動アドレス変換

自動アドレス変換機構の目的はダイナミックリロケーションを可能にするためのもので，プログラムの論理アドレスを物理アドレスに変換する機能を有する。シミュレータにおける自動アドレス変換は命令の実行を機械に直接行わせるため，命令のオペランドアドレスを自己が置かれているシミュレータ内のデータ領域のアドレスに変換する機能である。この他に共通情報を指すアドレスに対しても同様の変換操作を施す。

(5) タイマ

シミュレータにおけるタイマは1命令実行ごとに公称の実行時間をカウントするもので，プログラムの実行時間の測定およびプログラムがループに入つたときの指定時間後の打切り処理の手段にも使われる。

4.3.5 スーパーバイザ機能のシミュレーション

スーパーバイザの機能はタスクの制御，割込み処理，ＳＶＣマクロ

命令の処理などであるが，ユーザープログラムに直接関係しハードウエア的な制御にからまない範囲として，このシミュレータではＳＶＣマクロ命令に対する処置のみを行なつている。

ＳＶＣマクロ命令は動作の性質上次の３種類に大別して扱う。

(1) シミュレートしなければ以後の実行が不可能となるもの，あるいは出現が頻繁でシミュレーションの対象とすることが望まれるものに対しては，各ＳＶＣマクロ命令ごとに専用の処理ルーチンで等価な動作を行なう。（ EXIO,CHAIN,LOAD,WAIT など）

(2) シミュレーションするだけでは意味を失うものについては No Operation とする。（ AVBL,UNAVBL,LOCK など）

(3) シミュレーションの範囲をこえてしまうもの，あるいはシミュレーションが困難で出現頻度が少いものに対しては，その旨表示してオペレータの指示を待つ。（ FORK,AISCAN など）

4.3.6 入出力動作のシミュレーション

プログラムのデバッギングの過程ではそのプログラムの中に含まれる入出力動作が重要な資料を提供してくれることになる。従つて入出力動作はできるだけ克明にシミュレートすることが望ましい。MELCOM−350／30 の入出力はプロセス入出力，ペリフエラル入出力，ディスク入出力の３種類に大別できる。これらの動作のシミュレーション法について以下に述べる。

(1) プロセス入出力

プロセス入出力装置は，主として外部の被制御装置に直結されているので，シミュレーション時にこれを直接駆動することは避けねばならない。このためにシミュレータはプロセス入出力装置の代替としてタイプライタを用い，入出力要求があれば自動的にこの代替装置に切り換えてオンライン制御への悪影響を防ぐ。

(2) ペリフエラル入出力

ペリフエラル入出力装置は多種で相互の互換性にも乏しいので，

異種の装置による代替は行わず，装置の種類，IOCB(入出力コントロール情報ブロック)，チャネルプログラムのチェック(アドレス変換も含む)のみを行なつて後は機械の機能にまかせる。

(3) ディスク入出力

ディスク内の幾つかの領域はオンラインデータ用として使用されているのが通例であるから，シミュレーション中のプログラムがディスク入出力を要求している場合，これを直接実行させると誤まつた書込み指定によりオンラインデータ領域の内容がこわされるおそれがある。一方，ディスクのアクセス領域のチェックによりオンラインデータ領域の使用を禁止すると安全にはなるが，オンラインデータ領域を意識的に使用するプログラムのシミュレートができなくなり，機能的に不満がある。

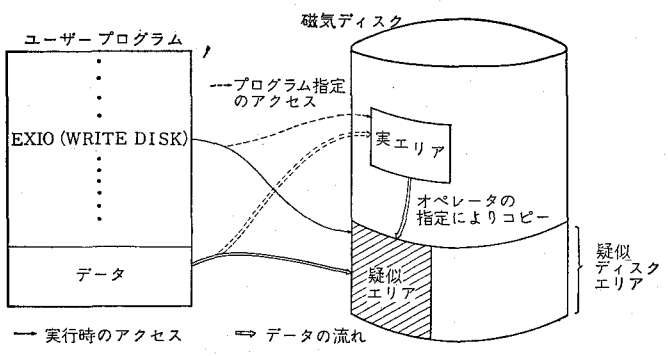


図 4.3 ディスクのシミュレーション

そこで，図 4.3 に示すようにディスク中の作業領域内に擬似ディスク領域を設け，アクセスしたい領域の内容をあらかじめこの擬似領域へ移しておく。シミュレータでは該当領域への入出力を検知すると自動的に擬似ディスク領域へのアクセスに切換えて実領域の内容がこわされるのを防いでいる。

オペレータが行なう操作は，プログラムでアクセスしているディスクの領域をコマンドで指定するぞけで，他の処置はシミュレータが自動的に行なう。これにより使用者にわずらわしさを感じさせずに，しかもディスクの任意の領域を自由にアクセスすることが実現できた。

4.3.7　共通情報のシミュレーション

(1)　スクラッチパッド領域のシミュレーション

主メモリの初番地の方にあるスクラッチパッド領域にはオペレーティングシステムに関する基本的なシステム定数が置かれており，読出し動作に限り一般のプログラムからも参照される。これに対するシミュレーション法としては，誤つて予定外のスクラッチパッドへアクセスすることを防止すること，および被シミュレートプログラムが異つた設置環境下で働くこともあることを考慮して，所要アドレスとその内容をあらかじめコマンドによつて与えてシミュレータ内に記憶しておき，スクラッチパッドのアクセスに際してはこちらの方を参照するやり方をとつている。

スクラッチパッド情報のうち共通情報領域の開始番地に対しては特にアクセスの頻度が高いので，シミュレータが自動的にこれを設定することにしオペレータの操作手数を省いている。

(2)　共通情報領域（GCA）のシミュレーション

共通情報領域ＧＣＡ（Global Common Area)はプログラム相互間のデータ授受のためにコア常駐領域に設けられているもので，オンライン用のプログラムとデータが含まれている。シミュレーション時におけるＧＣＡへの直接アクセスはディスクの場合と同様に避けねばならないから，図4.4に示すようにディスクの作業領域内に擬似ＧＣＡ領域を設けて実ＧＣＡの内容をそつくり移しておく。シミュレータはＧＣＡアクセスを要求する命令に対してディスク上の擬似ＧＣＡ領域から該

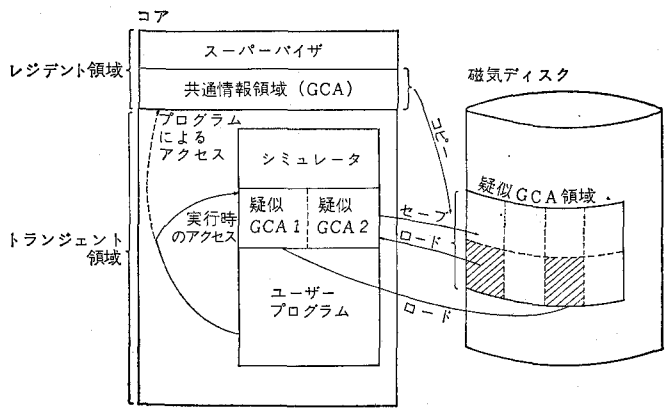


図4.4　共通情報のＧＣＡのシミュレーション

当するブロックをコア上の擬似ＧＣＡ領域へ読込んで，このブロック内の所要の部分をアクセスするように命令のオペランドアドレスをすりかえている。

別のブロックが新たに読込まれる時にはコアの擬似ＧＣＡ領域の内容はディスク上の擬似ＧＣＡ領域の元の位置に戻され，常に最新の情報を保持するようつとめる。またＧＣＡアクセス時のディスクの読み書き回数を減らすために，シミュレータ内の擬似ＧＣＡ領域を２個所設け，ディスクの擬似ＧＣＡから交互に読込むことによつて処理速度の向上を図つている。

以上の処理は実際にはシミュレータが自動的に行なうため，オペレータはＧＣＡアクセスに関してシミュレータ内の擬似ＧＣＡ領域の存在をまつたく意識する必要はない。

4.3.8　シミュレーション制御情報

シミュレータを動作させるために使用者が与える制御情報はコマンドの形をもつている。このコマンドはシミュレーション実施のための環境を作り出すことを主目的とするものと，デバッギングの制御命令として用いるものとに大別される。前節までに述べたシミュレーション機能と関連の強い前者の制御用コマンドを表4.2に示す。

表4.2　シミュレータ制御コマンド

コマンドの形式	機　　能
LPR, ××××××, {1 / 0}.	プログラムをロード 1：ロード後ホールト，0：ロード後実行
LPH, ××××××, {1 / 0}.	フェーズをロード 1：ロード後ホールト，0：ロード後実行
END.	シミュレータを QUIT
STA, ××××.	指定番地から実行開始
RST.	実行中断点から再開始
HLT.	SVC ホールトモードにセットキャンセルは HLT, —.
SKP.	SVC ホールト時に SVC マクロをスキップして実行再開
PGS, ××××, ××××.	GCA のアクセス可能領域を宣言
PDS, ××××, ××××.	プログラムでアクセスしているディスク領域を宣言，5領域まで可能 PDS, ———……. は一の数だけ宣言を取消す
SCR, ××××, {×××× / ＊}.	スクラッチパッドアクセスを許す宣言．＊はその時点での実スクラッチパッドの内容をセット
DAL, ××××.	ディスク上の疑似 GCA 領域のサイズをセットする．PGS 宣言はこの範囲内であること
STW, {CR / PT}.	コマンド入力装置をカードリーダ(CR)または紙テープリーダ(PTR)にセット STW, —・はタイプライタ
SHC, {LP / TW}.	ハードコピー装置をラインプリンタ (LP) またはタイプライタ (TW) にセット
LOG, {ON / OFF}.	コマンド入力装置が CR または PTR のとき，入力コマンドの印字指定 ON：印字，OFF：印字なし

4.3.9 デバッギング機能

プログラム・デバッギングの難易は計算機の使用効率に影響するとともに，プログラム開発速度に大きな関連をもつ。このオンラインシミュレータではデバッギングの能率化が大目標である。デバッギング機能に対する要求は次のような点である。

(1) プログラム実行時のあらゆる履歴が残せること

(2) データは10進数，16進数，浮動小数点数のどの形式でも入力可能なこと

(3) 出力情報はできるだけ関連する情報も含めて出して少ない操作回数で誤りを発見できること

(4) データ印字装置がタイプライタの場合，冗長な情報の出力を抑え速度を上げることが可能なこと

(5) 入力情報は厳密にチェックされ，操作ミスにより他のプログラムへの悪影響を与えないこと

(6) 入出力装置の切換えが自由にできること

(7) 必要ならスタックドジョブ形式でもデバッギングが進められること

(8) シミュレーション実行中任意の時点でオペレータが介入できること

このような諸点を考慮してこのシミュレータに設けられたデバッギング機能を実現するコマンドを表4.3に示す。

表4.3 デバグ用コマンド

コマンドの形式	機能
AHn, ××××. AHn, —. AH, —.	指定された番地で実行ホールト n=1, 2, 3 取消しは AHn, —. または AH, —.
TRn, ××××, ××××. TRn, —. TR, —.	指定範囲内にある命令を実行したとき，レジスタの内容をダンプ n=1, 2
BTR, ××××, ××××. BTR, —.	指定範囲内にあるブランチ命令を実行したとき，レジスタの内容をダンプ
MTR, ××××, ××××. MTR, —.	オペランドアドレスが指定範囲内にあるストア命令を実行したとき，レジスタの内容をダンプ
DXn, ××××, ××××, ××××. DFn, ××××, ××××, ××××. DXn, —. DX, —. DFn, —. DF, —.	指定番地の命令を実行時に指定領域の内容をダンプ. n=1, 2 DX は16進表示，DF は浮動小数点形式
CSW	デバッグ用の全スイッチをリセット
CCS. ××××.	コンディションコードをセット
RGa, ××××. RGX, ××××××××. RGF, ××××××××××.	各レジスタにデータをセット a=A, E, 1, 2, 3 RGX=Hexa., RGF=Float. でFAレジスタにセット
RDP.	全レジスタの内容をダンプ
CSa, ××××, ××……×, ……, ××……×.	指定コアメモリにデータをセット a=X, F, D X : Hexa., F : Float., D : Decim.
CDa, ××××, ××××.	指定コア領域の内容をダンプ a=X. F. D X : Hexa., F : Float., D : Decim.
DMS, ××××, ××××, ……, ××××.	指定ディスクメモリにデータをセット
DDP, ××××, ××××.	指定ディスク領域をダンプ
TTC, ××××, ×, ××××, ××××.	指定ディスク領域の内容を指定コア領域へ転送
TTD, ××××, ×, ××××, ××××.	指定コア領域の内容を指定ディスク領域へ転送
TIM.	プログラムの実行時間を表示
LTS, ×××××.	実行打切り時間をセット
パネルトグルスイッチ全ON	直後のブランチ命令を実行後，オペレータモードとなる

4.3.10　シミュレータプログラムの構造

(1)　プログラムの構造

多数のオンラインプログラムと同時実行するためにシミュレータの占有するコアメモリは少い程よい。一方被シミュレートプログラムを収容する領域すなわちシミュレータのデータ領域は大きい程よい。この相反する要求をできるだけみたすために，シミュレータプログラムは図4.5に示すようにレジデント部分とトランジエント部分とから成る構成をとつた。

トランジエント部分には処理機能別にまとめられたサブルーチンが必要に応じて呼び込まれる。レジデント部分には各トランジエントルーチン間で共通に使用されるデータおよびプログラムが置かれている。

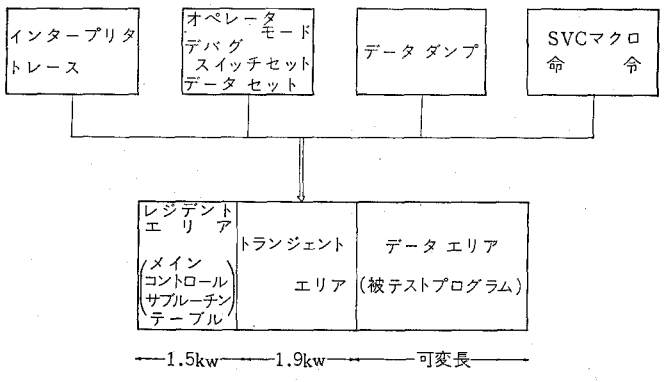


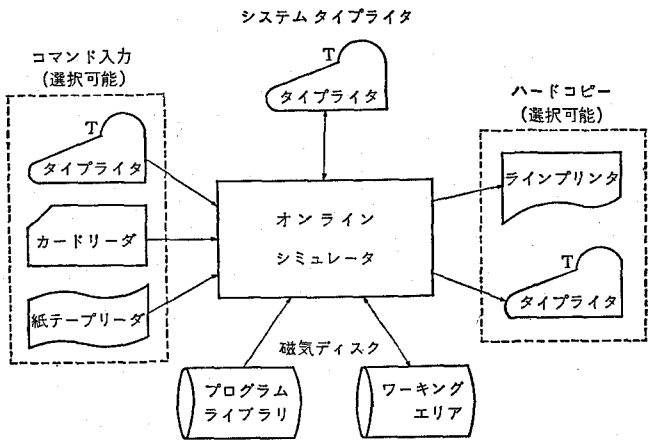
図4.5　シミュレータプログラムの構成

コアメモリ内にとるシミュレータプログラム領域は図4.5に示すように合計3.4 kWとなる。これとデータ領域を合わせたものがシミュレーションに必要なメモリ領域であり，データ領域内に入る被テストプログラムの大きさによつて異るが，現実にはテストされるプログラムの大きさが2 kW程度の場合が多いので，アセンブラなどのシステムプログラムが必要とする領域と同じ6 kWでシミュレーションを行なえる。コアの使用条件さえ許せば更に大きい領域を割当てることも可能である。

シミュレータプログラムそのものは全体で約8 kWの大きさとなつている。

(2)　シミュレータに必要なハードウエア構成

シミュレータの使用に必要なハードウエア構成を図4.6に示す。



計算機の設置構成が多様なことを考慮して種々の入出力装置を選択使用することを許している。図からわかるように最小の場合は入出力装置としてシステムタイプライタだけでもよい。

図 4.6　シミュレータに必要なハードウエア構成

4.4　結　　言

従来，非常に困難であつたオンライン制御中の計算機でのオンライン・デバッギングを容易にしたオンラインシミュレータについて述べた。計算機制御の処理内容の質的向上およびデータ処理の高度化に伴つて，このようなオンラインデバッギングエイドの役割はますます重要になつてくる。ここに述べた方式は同一機種でプログラムの動作環境を変更することを主眼としてシミュレーションの機能をとりまとめた点に特色がある。

実例としてMELCOM－350／30 オンラインシミュレータをあげて種々の問題に対処するための機能について述べた。ここではいかなる条件でもプログラムが暴走したり，制御動作に悪影響を及ぼしたりしないことを第一義に考え，インタープリタ方式を採用した。また，デバッギングの能率向上を目指して強力なデバッグ機能を豊富に持たせたので，オンラインシステムに限らず，計算センタなどのオンラインシステムでも有効な道具として使用されている。

このシミュレータの問題点はアセンブラ言語のプログラムを対象としていることで，その限りでは充分強力であるが，高位言語たとえば前述のCONFORM のプログラムを対象とする場合もオブジェクトプログラムのレベルで操作を考えねばならない。問題向き言語システムMDSSではソー

ス言語レベルでのデバッギングを可能としているが，手続き向き言語でそのような方式を実用化することが今後の問題である。

第5章　結　　論

プロセス制御応用で計算機言語システムに要求される事項を整理し，手続き向き言語および問題向き言語についてそれらの要求をみたす上での方法を実例を以つて示した。両者を含めた複合要素の組合せにより制御用プログラミングシステムを構成することが実用的であり，その場合にオブジエクトプログラムのレベルでプログラム構造上の一体化が可能となるよう各言語の機能を設計することが重要である。

問題向き言語システムは実行時の制御の補助のためのサブシステムを設ける場合が多く，メモリ効率および速度の点で問題となり得るがが，教育，ドキュメンテーションなどの関連で利点が大きいため，今後さらに有用となるであろう。プロセス制御応用に限らず，広く各種の分野に適用される可能性がある。

プログラムデバッギングはどんな応用分野でも必要であるが，特に制御応用ではプラント据付調整時にソフトウエアも含めてシステムチューニングが必ず行われるため汎用とは異つた意味をもつ。第4章ではそのためのシミュレータについて述べ，第3章では問題向き言語システムでソース言語レベルからのデバッギング操作が実現できることを示した。手続き向き言語システムで有効に使えるデバッギングツールの手法を開発することが残された問題である。

〔第Ⅲ編　参考文献〕

(1) Jarvis, P.H.：〝Some Experiences with Process Control Languages〟, IEEE Trans., IECI−15, P.54（1968−12）

(2) Roberts, B.C.：〝FORTRAN Ⅳ in a Process Control Environment〟, IEEE Trans., IECI−12, P.61（1968−12）

(3) Mohmeyer, R.E.：〝CDC 1700 FORTRAN for Process Control〟, IEEE Trans., IECI−15, P.67（1968−12）

(4) Bates, D.G.：〝PROSPRO／1800〟, IEEE Trans., IECI−15, P.70（1968−12）

(5) Weiss, E.A.：〝Conflict rages over Process Language Standards〟, Control Engineering, July 1969, P.77

(6) Pike, H.E.：〝Process Control Software〟, Proc.IEEE, 58−1（1970−01）

(7) Kelly, E.A.：〝FORTRAN in Process Control：Standardizing Extensions〟, Instrumentation Technology. 17−5, P.47（1970−05）

(8) 〝Minutes, Workshop on Standardization of Industrial Computer Languages〟, at Purdue University, 1st(Feb.1969)～8th（Oct.1972）

(9) 有田，首藤，関本，居原田：〝プロセス制御用プログラミング・システムの構成〟, 昭45信学全大，941（昭45）

(10) 有田，首藤，関本，居原田：〝プロセス制御用基本コンパイラ言語〟, 昭45信学全大，942（昭45）

(11) Brandes, J., Eichentopf, S., Elzer, P., Frevert, L., Haase, V., Mittendorf, H., Müller, G., Rieder, P.：〝PEARL the Concept of a Process- and Experiment-oriented Programming Language〟, Electronische Datenverarbeitung, Heft 10／1970

(12) 有田，首藤，関本，居原田：〝プロセス制御用コンパイラCONFORM〟，三菱電機技報，45－2，P.213（昭46－02）

(13) 三菱電機（株）：〝MELCOM 350－30／30F CONFORM説明書，SM－60143－A〟，（1971）

(14) 上滝：〝プロセス制御におけるプログラム言語〟，昭46電気学会全大，S．10－1

(15) 日本電子工業振興協会：〝工業用コンピュータにおけるソフトウエアの動向〟，46－A－52（昭46－03）

(16) 首藤，関本，春原：〝制御用手続き向き言語のための機能について〟，昭46情処学会大会，70（昭46－12）

(17) 上滝：〝工業用計算機言語とその標準化〟，電学誌，92－2，P158（昭47－02）

(18) 日本電子工業振興協会：〝工業用コンピュータソフトウエアの標準化動向（第一報）〟，47－A－56（昭47－03）

(19) 春原，太細，定松，加賀美，関本，首藤：〝プロセス制御用言語 CONFORM Ⅳ〟，昭47電気関係学会関西連大，G7－34（昭47－10）

(20) 春原，太細，定松，加賀美，関本，首藤：〝プロセス制御用言語CONFORM Ⅳコンパイラ〟，情処学会第13会大会，29（昭47－12）

(21) 三菱電機（株）：〝MELCOM 350－30／30F CONFORM Ⅳ説明書，RM－30023〟，（1972）

(22) 首藤，居原田：〝DDC言語とそのシステム構成の一方法〟，昭46情処学会大会，71（昭46－12）

(23) 首藤，五十嵐，居原田：〝プロセス制御用プログラミング・システム――MDSS――〟，昭47信学全大，1196（昭47－04）

(24) 居原田，五十嵐，首藤：〝ソース言語レベルでのオンラインプログラムのチューニングについて〟，情処学会第13.回大会，31（昭47－12）

(25) 中平，五十嵐，居原田，首藤：〝プロセス・データ・モニタリングにおけるファイルの一構成〟，情処学会第13回大会，126（昭47－12）

(26) 有田，井上，黒田，居原田：〝MELCOM－350／30 オンラインデバギン

グエイド〃，昭44信大全大，900

㉗ 有田，長谷川，今藤，居原田：〃オンラインプログラムシミュレーションの二，三の問題〃，昭44電気四学会関西連大，G7－3

㉘ 有田，井上，首藤，居原田：〃MELCOM－350／30 オンラインシミュレータ〃，三菱電機技報，44－5，P.664（昭45－05）

㉙ Inoue，Arita，Imafuji，Hasegawa，Suds，Iharada：〃MELCOM－350／30 On-line Simulation System〃，MITSUBISHI DENKI LABORATORY REPORTS，11－1／2，P－89

結　　論

計算機のプログラミング言語とコンパイラの分野について，言語系の構成の問題，コンパイラおよび実行時処理系の構成の問題，および処理系開発過程の効率化の問題を対象として，より高能率のシステムを実現する立場で行なつた研究の結果をまとめた。

各編について得られた結果は本文中に述べたので繰り返さないことにし，ここでは今後に残された問題について触れる。

マイクロプログラミングによるオブジエクト計算機およびコンパイラ機能の構成で論じた方法は，それぞれ類似の問題への適用が可能であるが，これらは計算機の機能単位のマイクロ化に通じるものである。高密度集積回路などの発達により，ハードウエアコストが低下するに従つて，従来ソフトウエアで実現していた機能をハードウエア化する傾向が現われているが，その場合にマイクロプログラミング技術が利用されることが多くなろう。この方法は本論文で対象としたような言語処理系ばかりでなく，オペレーテイングシステムの各部機能にも適用されるとみられるが，複雑に組み立てられている現在のオペレーテイングシステムを解きほぐして，基本的なマクロ機能群からなる体系に整理することが課題である。

計算機は規模の小さいものほど用途別に性格を明確にする傾向があり，超小形事務用計算機がミニコンピュータと共存しているが，生産者にとつては低価格のものほど共通化してコスト低下をはかりたい事情がある。この問題の解決策としてもマイクロプログラミング技術が期待され，ほぼ共通の中央処理装置に対して，入出力機器の接続とマイクロプログラミングによる10進処理系など各種処理機能の付与を行なつて幾つかの用途，分野に適合する機種を実現することが考えられる。この場合，小形であるほどマイクロプログラミングを低コストで実現することが課題となる。

オンライン制御用計算機では，その環境の要請により独特の技術が発達してきたが，汎用計算機分野においてリアルタイム処理が普及するにつれて，制御

用の技術がとり入れられ一般化しつゝある。また逆に制御用計算機応用でも局部的な制御から，全体的なシステムへと対象が拡大し，工場の生産活動に直結するデータ処理までを含める方向にある。これにより両者の差は次第になくなりつゝあり，大形の計算機ほど共通的となつてきている。ソフトウエアについても，汎用，制御用の要求に綜合的に応じ得るよう，体系化が進められることが予想される。

プログラムシステムの開発効率化については，その道具となるシステム作成用言語が実際的な対策として要求される一方，言語処理システム等では，言語および翻訳に関する理論に基礎を置いた諸方法の試みをはじめとする地道な技術の積上げがなお当分必要であろう。

謝　　辞

本研究の過程において終始客観的な立場からの適切な御助言と親身な御鞭撻をいただき，研究をまとめるにあたり懇切なる御指導と励ましをいただいた大阪大学工学部電子工学教室・尾崎弘教授に衷心より御礼申し上げる。

本研究をまとめる過程で種々な御指導と御鞭撻をいただいた大阪大学工学部精密工学教室・牧之内三郎教授，電子工学教室・喜田村善一教授，ならびに通信工学教室・板倉清保教授に対して心から深謝申し上げる。

研究の過程において終始適切な御助言と励ましをいただいた大阪大学基礎工学部情報工学教室・嵩忠雄教授ならびにその研究室の方々に心から感謝の意を表する。

またとくに第Ⅲ編に関して，応用問題として見過されがちなこの種の研究の実用面での評価を通じて，研究推進上の御助言と御鞭撻をいただいた電子技術総合研究所制御部長・上滝致孝博士に深謝申し上げる。

本研究は著者が三菱電機株式会社中央研究所および鎌倉製作所において行なつたものであり，数年間にわたり所属を変えつゝもほゞ一貫して同一分野の研究に従事できたこと，常に上長の御理解と御援助を得たことが研究遂行の大きな力となつている。現・同社情報システム部長・豊田準三博士，現・北伊丹製作所長・喜連川隆博士，現・鎌倉製作所計算機工場計算機製造部長・大鳥羽幸太郎氏，鎌倉製作所電子機器研究部長・中村敏行氏，現・物流推進本部長付・吉江高明氏，鎌倉製作所計算機工場ソフトウエア技術部次長・嶋村和也氏，同所電子機器研究部主任研究員・有田不二男博士ならびに同所電子機器研究部主任研究員・磯崎真氏に対し厚く感謝の意を表する。

研究の過程では多くの同僚および関係者の御協力を得た。特に直接に協力を得た関本彰次氏，魚田勝臣氏，居原田邦男氏，ならびに小碇暉雄氏に対し謝意を表する。

付　録

著者　発表論文目録

（＊印は関連発表論文）

（1） 八島，首藤，犬伏：〝原子力プラントの動特性解析への計数型電子計算機の応用〃，日本学術会議　第3回原子力シンポジウム報文集，1，V－14.（昭34）

（2） 首藤：〝微分解析ルーチンDASについて〃，数理科学総合研究第2，3，4班合同シンポジウム予稿，（昭35）

（3） 八島，首藤：〝計数形電子計算機による微分解析機のシミュレーション〃，昭35電四連大，381.（昭35－07）

（4） 穂坂，嶋村，中島，吉江，首藤：〝デイジタル演算高速化装置(1)〃，三菱電機，34－9，（昭35－09）

（5） 穂坂，嶋村，中島，吉江，首藤：〝デイジタル演算高速化装置(2)〃，三菱電機，34－10，（昭35－10）

（6） 豊田，吉江，首藤：〝電子計算機の工作機数値制御への応用〃，昭35電学関西連大，Ⅳ－5.（昭35）

（7） 豊田，中塚，吉江，前田，首藤，壺井，菅，魚田：〝計数形電子計算機の特殊演算高速化方式〃，三菱電機，34－11，（昭35－11）

（8） 豊田，中塚，吉江，首藤：〝遅延線方式による演算高速化〃，昭35情処学会全大，47.（昭35）

（9） 豊田，吉江，首藤：〝2進電子計算機における数変換の一手法〃，昭36電四連大，309.（昭36）

（10） 吉江，首藤，魚田：〝2進電子計算機における直並列論理回路による高速乗除算の一手法〃，昭36電四連大，310.（昭36）

（11） 吉江，首藤，魚田：〝2進電子計算機における浮動小数点演算の一手法〃，昭36電四連大，311.（昭36）

（12） 豊田，中塚，吉江，前田，首藤，壺井，菅，関本，魚田：〝計数形電子計算機MELCOM－LD1〃，三菱電機，35－5，P.860.（昭36－05）

(13) 首藤：〝ブロック図から直接コード化するプログラム方式〃，三菱電機 **35－6**，P.1021．(昭36－06)

(14) 菅，吉江，首藤，関本，魚田：〝MUSE Program Systemの概要〃，三菱電機，**35－8**，(昭36－08)

(15) 吉江，首藤：〝工作機の数値制御用プログラム方式〃，三菱電機，35－9，P.1335．(昭36－09)

(16) 吉江，首藤，田中：〝演算回路の動作試験〃，昭36情処学会全大，23．(昭36)

(17) 吉江，首藤，関本，木村，魚田：〝MAMA自動プログラム方式〃，三菱電機，**36－5**，(昭37－05)

(18) 吉江，首藤，関本，木村，魚田：〝記号化プログラムシステムMAMA〃，昭37電学関西連大，4－23．(昭37)

(19) 吉江，首藤，関本，魚田，木村：〝MUSEおよびMAMAシステム〃，昭38電四連大，456．(昭38－04)

(20) YOSHIE，SUDO，SEKIMOTO，KIMURA，MIURA：〝MAMA—— A Symbolized Program Compiling System〃，MITSUBISHI DENKI LABORATORY REPORTS，**4－3**，P.309．(July 1963)

(21) 吉江，首藤，関本，魚田，佐久間：〝MUSE自動プログラム方式〃，三菱電機，技報，**37－8**，P.1056(昭38－08)

(22) 首藤：〝コンパイラー作成上の問題〃，昭38電学関西連大，S11－4(昭38)

(23) YOSHIE，SUDO，UOTA，SAKUMA：〝MUSE—— An Algorithmic Language Compiling System〃，MITSUBISHI DENKI LABORATORY REPORTS，**5－1**，P.55(Jan. 1964)

(24) 首藤，三浦：〝計算機入出力装置のメモリ保護の一方式〃，昭39電四連大，311．(昭39)

*(25) 首藤，魚田，居原田：〝ALGOLコンパイラ向き基本言語試案〃，昭39信学全大，553。(昭39)

*(26) 首藤，関本，魚田，鶴岡：〝MELCOM－1530 COBOL システムの構

成〃，昭３９信学全大，５５４．（昭３９）

(27) 首藤，魚田，居原田：〃計算機基本言語に関する一考察〃，昭３９情処学会全大，３．（昭３９）

(28) 首藤，関本：〃ALGOL言語とそのCompiling手法〃，三菱電機技報，**３８－１２**，P.１７２８（昭３９－１２）

(29) 首藤，魚田，居原田：〃計算機基本言語に関する一考察〃，三菱電機技報，**３９－３**，P.４３１（昭４０－０３）

(30) 首藤，魚田：〃擬命令リストを用いた構文解析の一手法〃，昭４０電四連大，６３７．（昭４０）

*(31) 首藤，関本，鶴岡：〃MELCOM－１５３０ COBOLシステムに於ける言語変換機構とオブジエクトの構成〃，昭４０電四連大，６３８．（昭４０）

(32) 首藤，魚田：〃コンパイラにおける名札変換の応用〃，昭４０電四連大，６３９．（昭４０）

(33) 首藤，関本，吉竹：〃シンタツクスに基づくALGOLコンパイラとそれに関する諸問題〃，三菱電機技報，**３９－１１**，P.１３７４（昭４０－１１）

*(34) 首藤，魚田，居原田：〃計算機を用いたコンパイラ作成自動化の実験〃，第７回プログラミング・シンポジウム報告集，C－５３（昭４１－０１）

*(35) SUDO，UOTA，IHARADA：〃Some Considerations and Experiments on the Basic Programming Language〃，MITSUBISHI DENKI LABORATORY REPORTS，**７－１** P.４０（Jan.１９６６）

(36) 首藤，魚田，居原田：〃計算機を用いたコンパイラ作成自動化の実験〃，情処学会月例会資料１１（昭４１－０３）

*(37) 首藤，魚田：〃コンパイラ自動作成の一方法〃，信学誌，**５０－４**，P。６４９（昭４２－０４）

(38) 嶋村，首藤，藤井，中山：〃MELCOM－３１００ソフトウエア(1)――モデル１０プログラムの概要――〃，三菱電機技報，**４１－４** P.５３０（昭４２－０４）

(39) 首藤，関本，武田，三光：〃MELCOM－３１００ソフトウエア(2)――モデル３０オペレーテイングシステムの概要――〃，三菱電機技報，**４１－１０**，

P.1307（昭42－10）

(40) 首藤，魚田，斉藤：〝プログラム書きかえ問題へのCOBOLの応用〃，三菱電機技報，**41－11**，P.1425（昭42－11）

(41) 国分，有坂，首藤，魚田：〝事務用言語ACEコンパイラとその応用〃，昭42情処学会大会，3.（昭42）

(42) 首藤，中山，末沢，東海林：〝MELCOM－3100ソフトウエア(3)――モデル30Tにおけるプログラムの処理――〃，三菱電機技報，**42－3**，P.493（昭43－03）

(43) 首藤，野田，石川，長田：〝MELCOM－3100ソフトウエア(4)――10PTシステムの概要――〃，三菱電機技報，**42－4**，P.620（昭43－04）

(44) 首藤，魚田：〝事務処理用言語MELCOM－ACEとそのコンパイラ〃，昭43電四連大，2481.（昭43）

*(45) 国分，有坂，首藤，魚田：〝MELCOM－3100ソフトウエア(5)――ACEコンパイラシステムの概要――〃，三菱電機技報，**42－10**，P.1360（昭43－10）

(46) 首藤，魚田：〝プログラム書き換え問題に関する二，三の試み〃，情処学会コンパイラ自動作成シンポジウム，（昭44－08）

*(47) 有田，井上，首藤，居原田：〝MELCOM－350／30 オンラインシミュレータ〃，三菱電機技報，**44－5**，P.664（昭45－05）

*(48) 有田，首藤，関本，居原田：〝プロセス制御用プログラミング・システムの構成〃，昭45信学全大，941.（昭45）

*(49) 有田，首藤，関本，居原田：〝プロセス制御用基本コンパイラ言語〃，昭45信学全大，942.（昭45）

(50) 首藤，関本，春原，太細，定松：〝PL／I制御用サブセットとMELCOM－350を用いた実験コンパイラについて〃，昭45信学全大，（昭45）

(51) SUDO：〝Outline of an Experimental PL／I Compiler Implementation for a Medium Scaie Industrial Computer〃，Minutes 4th Workshop on Standardization of Industrial

Computer Languages, Purdue Univ., P.224（NOV.1970）

(52) INOUE, ARITA, IMAFUJI, HASEGAWA, SUDO, IHARADA：〝MELCOM−350／30 On-line Simulation System〃, MITSUBISHI DENKI LABORATORY REPORTS，11−1／2，P.89

(53) 首藤：〝制御用計算機のソフトウエア（言語）〃，計測制御学会制御用計算機講習会．（昭46−11）

*(54) 有田，首藤，関本，居原田：〝プロセス制御用コンパイラCONFORM〃，三菱電機技報，45−2，P.213（昭46−02）

(55) 首藤，関本，春原：〝制御用手続き向き言語のための機能について〃，昭46情処学会大会，70．（昭46−12）

(56) 首藤，居原田：〝DDC言語とそのシステム構成の一方法〃，昭46情処学会大会，71．（昭46−12）

(57) 首藤：〝工業用計算機言語の開発と標準化の動向〃，情処学会第13回プログラミング・シンポジウム，D1．（昭47−01）

*(58) 首藤，小碇：〝動的な表管理とCOBOLコンパイラへのその応用〃，情処学会誌，13−2，P.113．（昭47−02）

(59) 関本，向井，首藤：〝LR(k) PARSERの簡易化について〃，信学会オートマトン・インホメーション理論研究会資料，A71−116（昭47−03）

(60) 首藤，関本：〝LR(k)文法について簡易化されたParserについて〃，昭47信学全大，164．（昭47−04）

*(61) 首藤，五十嵐，居原田：〝プロセス制御用プログラミング。システム——MDSS——〃，昭47信学全大，1196．（昭47−04）

(62) 立花，首藤：〝システム分解によるDDCのフエイルセイフの復帰性と自動切替の改善〃，第11回計測制御学術講演会，3314．（昭47−08）

*(63) 居原田，首藤，立花：〝DDCにおけるPIDアルゴリズムの飽和特性についてについて〃，昭47電気関係学会関西連大，G7−33．（昭47−10）

(64) 春原，太細，定松，加賀美，関本，首藤：〝プロセス制御用言語CONFORM Ⅳ〃，昭47電気関係学会関西連大，G7−34．（昭47−10）

(65) 居原田，五十嵐，立花，首藤：〝DDC用ソフトウエアMDSSとその構

成〃，第１５回自動制御連合講演会，３０６６，（昭４７－１１）

(66) 亀井，立花，居原田，首藤：〃DDC言語MDSSによるボイラ昇温制御系の構成と適用〃，第１５回自動制御連合講演会，３０６１．（昭４７－１１）

(67) 春原，太細，定松，加賀美，関本，首藤：〃プロセス制御用言語 CONFORM Ⅳ コンパイラ〃，情処学会第１３回大会，２９．（昭４７－１２）

(68) 高野，関本，首藤：〃Grammar GによるCellular Space C(G)について〃情処学会第１３回大会，８０．（昭４７－１２）

(69) 向井，関本，首藤：〃プログラムのセマンテイクスについて〃，情処学会第１３回大会，８１．（昭４７－１２）

(70) 居原田，五十嵐，首藤：〃ソース言語レベルでのオンラインプログラムのチユーニングについて〃，情処学会第１３回大会，３１．（昭４７－１２）

(71) 中平，五十嵐，居原田，首藤：〃プロセス・データ・モニタリングにおけるファイルの一構成〃，情処学会第１３回大会，１２６．（昭４７－１２）

(72) 関本，向井，首藤：〃LR(k)Parser最小化問題について〃，信学論D投稿中