U.D.C. 681.3.06+.08]:[681.323:62-52]

計算機制御用プログラミングシステム
Programming System for Computer Control

This article introduces a non-process monitor system for compiling a control program for use with Hitachi controlling computer HIDIC series (HIDIC 150/350/500/700) and a support system—MCS (Man-Machine Communication System—to be used for tuning-debugging compiled programs. These systems are designed with emphasis on facility of composing on-line real time systems.

桑原　洋*　*Hiroshi Kuwahara*
林　利弘*　*Toshihiro Hayashi*
三巻達夫**　*Tatsuo Mitsumaki*
坂東忠秋***　*Tadaaki Bandou*

1 緒　言

最近の計算機制御システムの規模とその適用範囲は大きく拡張されてきており，一方ではシステム完成までの期間は技術革新のテンポに合わせて短くすることが要請されている。しかし規模が大きくなるほど，その仕様決定に時間がかかり，この要請を満足するためには仕様が決定した以降の主要な作業であるプログラミングをいかに能率よく短期間に行なうかがますます重要な課題となってくる。

特に，計算機制御用プログラミングシステムは，単に個々のプログラムを作成するという狭義のものだけでは不足で，個々に作られたプログラムを複数個結合し，さらにオンラインシステムとして組み上げるまでのすべての過程を能率よくサポートするものでなければならない。以下，このような観点にたって，HIDICシリーズのプログラミングシステムの全容について述べる。

なお，**図1**はHIDICシリーズのオペレーティングシステムにおけるプログラミングシステムの位置づけを示すものである。

2 制御用プログラミングおよびプログラミングシステムの特徴

制御用プログラミング作業の大きな特徴として下記が考えられる。

(1) 制御用計算機はプロセスデータ，マンマシンデータをオンラインリアルタイムに扱うため，この処理特有の言語表現が必要である。

(2) 個々に作られたプログラムを結合し，さらにそれらのプログラムをオンラインシステムとして組み上げる作業が必要である。

(3) オンラインシステムとして組み上げられたプログラムのチューニング作業が必要である。

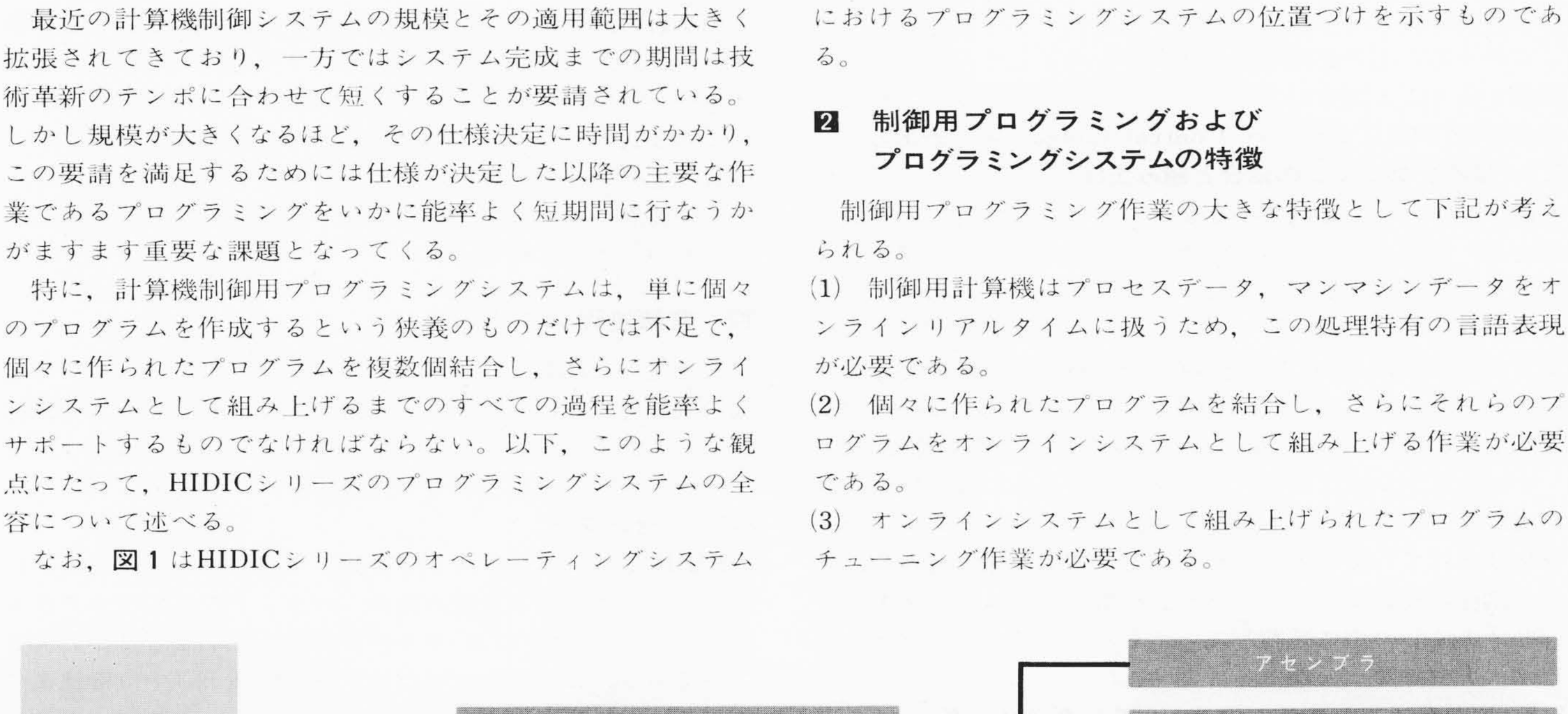

図1　HIDICシリーズオペレーティングシステム構成図　HIDICシリーズのオペレーティングシステムはTSESとして全体が統括されており，その中にはプログラム作成のためのNPMSとタスクの動きをモニタするPMSから成り，プログラミングシステムの大部分はNPMSに属している。

Fig. 1　Skelton Diagram of HIDIC Series' Operating System.

* 日立製作所大みか工場　** 日立製作所中央研究所 工学博士　*** 日立製作所日立研究所

以下，これらのおのおのについてその詳細を述べる。

2.1　制御用として必要な言語仕様

制御用として要求される言語仕様は，

(1)　必須項目として，

(a)　ビット単位のデータが扱えること。

(b)　各種のオンライン用入出力機器が扱えること。

(c)　システム共通エリアを参照できること。

があり，

(2)　あれば能率よい項目として，

(a)　構造形式データの記述

(b)　コアメモリと補助メモリを仮想的に同一データ空間として扱えること。

(c)　各種のI/Oに対する書式制御機能

(d)　言語レベルでのマクロな各種デバッグ機能

がある。

このうち，(1)の必須項目を満たすものとして，アセンブラ言語があり(1)と(2)のいずれをも満たすものとして，プロセス制御言語(PCL：Process Control Language)[1]がある。

2.2　複数プログラムの結合と組み上げ

個々に作られた比較的小さなプログラム(デック)を複数個結合し，全体として一つの大きな機能を持つプログラム（タスク）に成長させるものとしてバインダという言語サポートプログラムがある。ここでデックとはHIDICプログラミングシステムNPMS(Non Process Monitor System)の最小処理単位であり，タスクとはHIDICオンライリアルタイムモニタシステムPMS(Process Monitor System)の処理単位である。

オンラインシステムは，このタスクがさらに数十個から数百個協調しあって動くものであり，このオンラインシステムを作り上げていくためにこれらのタスク間の相互関連や，メモリ割付けをしながらタスクを計算機システム（主メモリと補助メモリ）の中に装荷していくものとして，ローダというサポートプログラムがある。

2.3　オンラインシステムのチューニングデバッグ

オンラインシステムとして組み上げられたシステムを運転調整しながら完全なものとしていく作業がいわゆるチューニングデバッグであるが，このとき個々のタスクやデータエリアの内容を見たり修正したり，またプログラムの特定の場所でその実行をとめたりする必要にせまられる。これらの作業を能率よく，かつ操作ミスを少なくするようにするため，MCS(Man Machine Communication System)というサポートプログラムがある。

このようにして組み上げられたオンラインシステムは，その後プラントの増設あるいは機能の拡張という以外は変更されないので，一般にこれらのシステムを収容するハードウェアは単なる「入れもの」と考えられ，機能を満足するのに必要なものだけあればよいとされる。このためどうしてもハードウェアにかける費用は制限され，個々のプログラムもコンパクトに作ることが要求される。このプログラムのコンパクト性に特に重要な影響を与えるのが，言語処理部である。言語処理部としてはアセンブラとコンパイラがあり，アセンブラは作成者の能力さえあればハードウェアの能力をぎりぎりまで生かした最適なプログラムを作ることができ，コンパイラは作成者の能力にも依存するが，それ以上にコンパイラの性能に大きく左右される。HIDICシリーズに用意されているPCLコンパイラはこの点に特に留意して作ったものであり，その性能は従来のコンパイラに比べて格段の進歩をとげている。なおプログラミング言語として，アセンブラを選ぶか，コンパイラを選ぶかコスト面からの目安は下記の式によって与えられる。

アセンブラを用いた場合のプログラム作成費用をP_A，所要メモリ費用をM_A，とし，コンパイラを用いた場合のプログラム作成費用をP_C，所要メモリ費用をM_Cとした場合，

$P_A+M_A>P_C+M_C$……(1)　ならば，コンパイラ

$P_A+M_A<P_C+M_C$……(2)　ならば，アセンブラ

をそれぞれ選ぶのがよい。

PCLコンパイラの場合M_C/M_A ＝1.15～1.25であることが確認されているので，最悪値をとって$M_C/M_A=1.25$として上記の1式に代入するとコンパイラを選ぶ基準は，

$$P_A-P_C>\frac{M_A}{4} \quad \cdots\cdots(3)$$

となる。

すなわち，アセンブラによるプログラミング費用とPCLによるプログラミング費用の差が，アセンブラを用いたときに必要とするメモリ費用の¼を越えれば，PCLのほうが有利といえる。

この3式の目安は，プログラミング費用の大半を占める人件費の上昇とメモリ費用の技術進歩による低下を考えると，その平衡点はますます規模の小さい計算機の分野に移行していくものと考えられる。

3　言語処理システム[2]

2.で述べたHIDICシリーズの各種言語処理関係プログラム間の処理の流れは，**図2**に示すとおりである。以下，HIDICシリーズ言語処理システムを構成する各プログラムについてその概要，特長，などについて述べる。

3.1　アセンブラ

HIDICシリーズのアセンブラは，アセンブラ言語で書かれたソースプログラムを読み込み，バインダの入力プログラム(バインダ語プログラム)をそのオブジェクトとして出力するものである。このアセンブラは，下位機種から上位機種まですべて同一思想で作られており，いずれもソースプログラムを2回調べるいわゆる2パス方式をとっている。すなわち第1回めはソースプログラムを読み込んでラベルのアドレス付けをし，第2回めでは再びソースプログラムを読み，第2回めの情報を用いてオブジェクトプログラムを作成するものである。この方式はオブジェクトプログラムにいっさいのむだな命令やデータがはいらず，コンパクトにできるので制御用としては最適である。

一般にこの方式は第2回めの処理に備えて，ソースプログラムおよび第1回めの処理結果を覚えるためにかなりの記憶容量を必要とするが，これを避けるためHIDIC 150のような小形機種では第1回めの処理結果のみを記憶しておき，ソースプログラムは計算機の中に記憶せず2回計算機に読み込ませるようにしている。

アセンブラ言語は，機械命令とアセンブラ命令からできており前者はハードウェアの命令と1：1に対応しており，後者は定数定義命令やエリア確保命令，エリア定義命令などの宣言命令とサブルーチンリンクのためのコール命令とから成っている。これらの命令においては，下位機種の命令は上位機種のそれのサブセットとなっており，特に機械命令についてはいずれもたとえハードウェアにない命令を書いてもアセンブルし，実行時には疑似命令としてソフトウェア的に処理できるようになっている。

3.2　コンパイラ

HIDICシリーズではJISに準拠したFORTRANコンパイラ

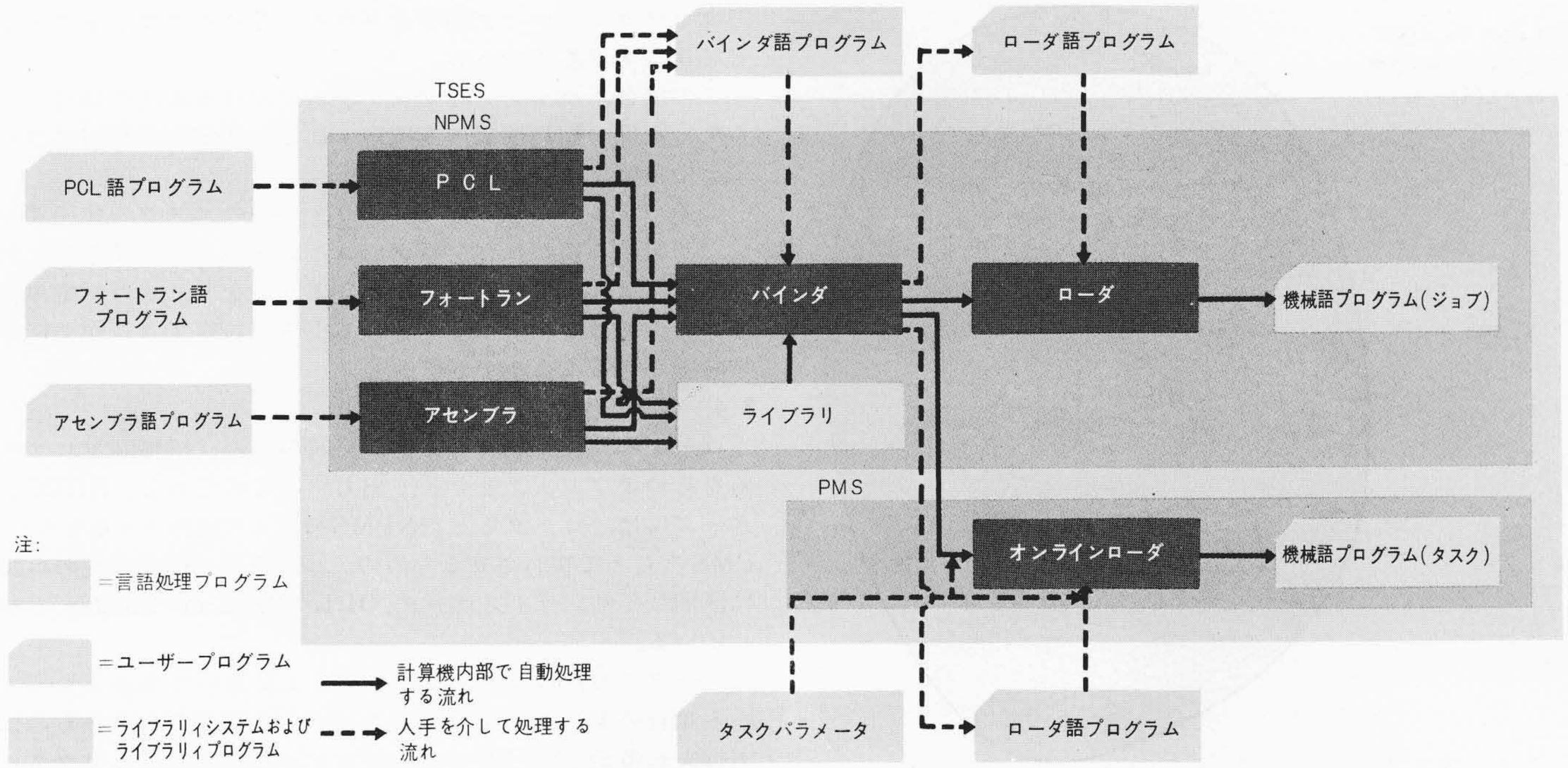

図2　HIDICシリーズにおける言語処理の流れ　PCL語プログラムはPCLコンパイラにより，フォートラン語プログラムはフォートランコンパイラにより，またアセンブラ語プログラムはアセンブラにより，それぞれ変換されて同一形式のバインダ語プログラムになり，バインダにより自由に結合される。結合されたプログラムはローダにより機械語に変換されて実行される。

Fig. 2　Skeleton Flow of Language Processing in HIDIC Computers

およびFORTRANをベースにPL/I的要素を加え，さらに制御用として必要十分な機能と性能を備えた，制御用言語PCLの二つを用意している。

(1)　FORTRAN

HIDICシリーズのFORTRANコンパイラは，FORTRAN言語で書かれたソースプログラムを読み込み，バインダの入力プログラム(バインダ語プログラム)をそのオブジェクトとして出力するものであり，HIDIC 500に対してはコア16kWドラム128kWで動くJIS5000レベルのものが用意されている。

(2)　PCL

PCLコンパイラはPCL言語で書かれたソースプログラムを読み込み，バインダの入力プログラム(バインダ語プログラム)をそのオブジェクトとして出力するものであり，オンラインリアルタイム用プログラムのすべてを，コンパイラ言語で作ることを目的として開発されたものである。このPCLはHIDIC 150，同350，同500，同700に対して使用でき，言語形式としてはFORTRANをベースにしている。従来，オンラインリアルタイムシステムといえば，アセンブラというのが常識であったが，これは従来のFORTRANが主として科学技術用に作られており，オンラインリアルタイムシステム用のプログラムを表現する機能がなく，無理に表現しようとすると作業能率，実行能率のいずれもが非常に悪くなり実用的でなかったためである。

以下にPCLの言語上の特長について述べる。[3][4]

(1)　整数形データを使いやすくした宣言文

IMPLICIT文の導入によりI·J·K·L·M·N以外で始まる文字に対しても暗黙的な整数形の扱いができる。

(2)　構造指定による階層形式データの記述

特に数ワードから成るテーブルの中を数ビットずつに分割しそのおのおのに名前をつけ，その名前を用いて各種の演算が行える。また同種のテーブルについては全体を一つの集合としても扱える。

(3)　豊富なビット処理機能

BIT DIMENSION文，BIT DATA文，DECISION IF文，DECISION DATA文などのビットに関する豊富な機能がある。

(4)　タスク間にまたがるデータに対する宣言文

主メモリ上のタスク間共通データに対してはGLOBAL文，補助メモリ上のタスク間共通データに対してはBULK文がある。

(5)　仮想メモリの概念を持つデータ

FORTRANでは主メモリ上のデータのみが，その演算対象であったが，PCLでは**図3**に示すように補助メモリ上のデータも主メモリ上のデータと全く同様に扱え，特別な入出力処理をユーザーは考える必要がない。

(6)　システム交信文の組込み

PMSとの交信命令を使いやすくするために，ステートメントとして組み込んである。

(7)　豊富な拡張性のある入出力機能

C/R, L/Pなどの一般I/Oはもとより，AI, DI, AO, DOなどのプロセスI/Oやカラー CRT(ブラウン管ディスプレイ装置)までもが同一形態のステートメントで簡単に扱える。

(8)　アセンブラ言語で書かれた既製プログラムとの容易な結合性

一般にアセンブラとコンパイラではその引数処理に違いがあるが，PCLではどのような引数処理の行なわれているプログラムに対しても容易に結合ができる。

(9)　豊富なデバッグユーティリティ

言語レベルでのダンプやトレース機能を行なう，DEBUG VALUES文，DEBUG FLOW文，オフライン処理とオンライン処理のステートメントを切り換えるCOMPILE MODE文，使用サブルーチンを切り換えるRENAME文，また配列

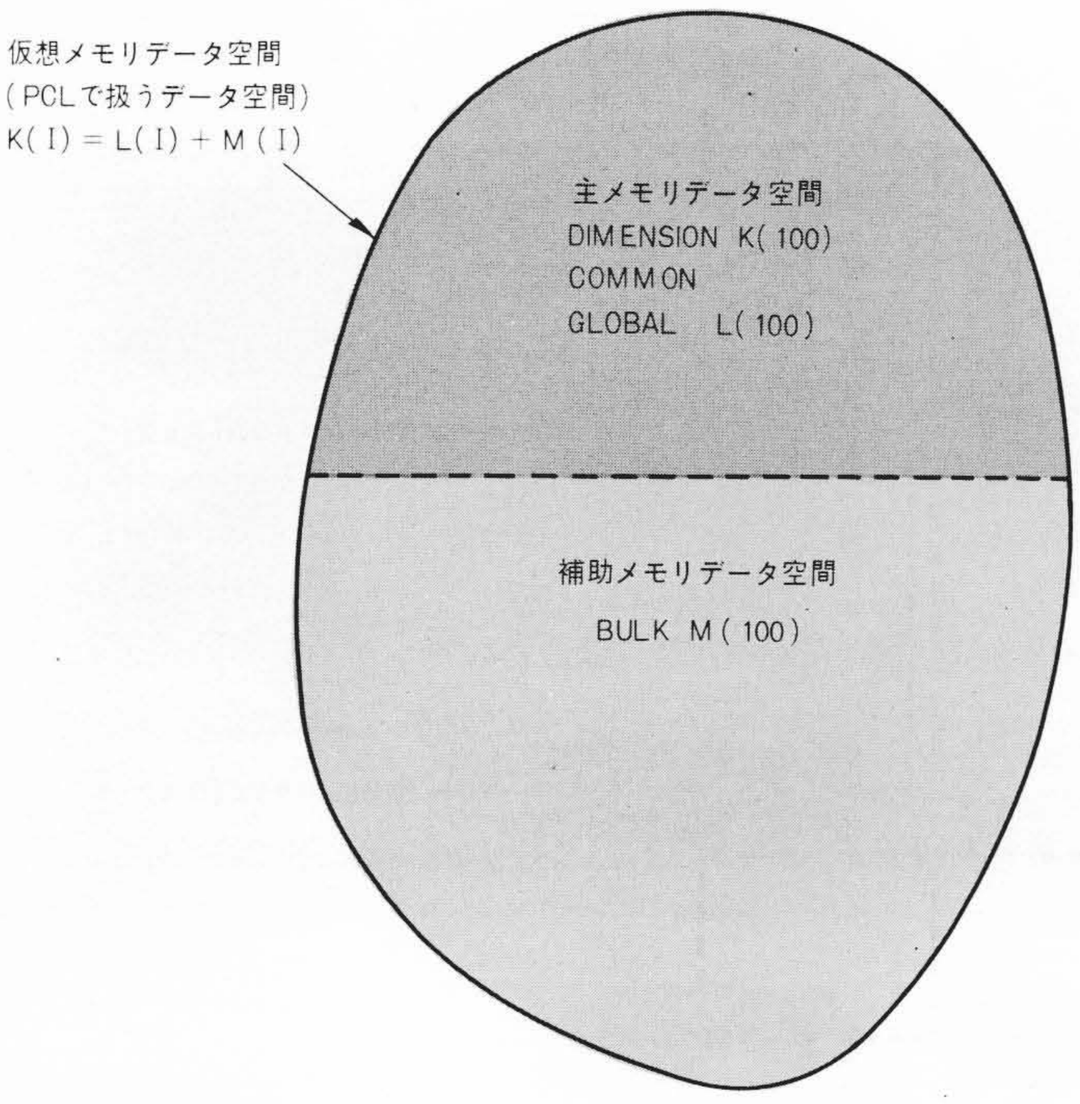

図3　仮想メモリデータ空間の概念図　主メモリと補助メモリを区別せずに，全体を一つのメモリ(仮想メモリ)空間と考えてデータの処理ができる。
Fig.3　Concept of Virtual Memory Space

添字のオーバーフローチェックを行なうDEBUG SUBSCRIPT文などがある。

上記，特長のうち使いやすさという点から特に画期的なのは(5)の仮想メモリデータ処理であり，これにより従来よく行なわれていた，システムエンジニアやプラントエンジニアが作成したプログラムを計算機のメモリ構成にあわせて作り直すという作業はなくなるとともに，一般のプログラミング作業においても大幅な労力削減が図れる。[5]また，PCL コンパイラとしては，下記のものが大きな特長としてあげられる。

(1)　16ビットマシンを生かした効率よいオブジェクト

HIDICシリーズのハードウェア命令は，すべて短語形式(16ビット命令)と長語形式(32ビット命令)を持っているが，できるだけ短語形式で済むようにオブジェクトを生成している。

(2)　補助メモリの効率よい利用を目ざしたオブジェクト

初期値を持たない配列エリアは，本来ワークエリアと考えられ，これは実行時にのみ主メモリ上にあればよいものなので，補助メモリには確保しないようにオブジェクトを生成している。

(3)　入出力処理プログラムのタスク化

入出力処理プログラムを独立したタスクにすることにより，CPU(中央処理装置)と入出力機器の並列処理を行なわせ，同時に主メモリに常駐化することなく複数のタスクから共用できるようにした。また，入出力データの受授をバッファリングすることにより，低速入出力機に対する一時的な高トラフィック処理要求を平滑化している。

なお，PCLコンパイラは，HIDIC 500/700についてはターゲット計算機を用いてコンパイルでき，HIDIC 150/350についてはホスト計算機(HITAC 7250，HIDIC 700/500)を用いてコンパイルができるようになっている。

3.3　バインダ

バインダはアセンブラやコンパイラのオブジェクトであるバインダ語プログラムを複数個読み込み，1個のローダの入力プログラム（ローダ語プログラム）をそのオブジェクトとして出力するものである。

この場合，バインダ語プログラムにおける未決の項目のうちデック間の相対位置のみ確定すれば，決定する項目についてその情報を作り出す。

このバインダがあることにより，一つのタスクの中でアセンブラ言語で書かれたプログラムとコンパイラ言語で書かれたプログラムの混合使用ができるようになり，また作業単位をタスクではなく，デックとして小さく設定でき，作業能率を向上させることができる。

3.4　ローダ

ローダはローダ語プログラムを読み込み，機械語プログラムをそのオブジェクトとして出力するものである。HIDICシリーズには，ローダとしてNPMSのもとで実行させるものと，PMSのもとで実行させるものの二つがあり，後者のものは特に区別してオンラインローダ(OLL：On Line Loader)と称している。

ローダの役割はローダ語プログラムにおいて未決となっている項目のすべてを決定することであり，そのおもなものは実行先頭番地を作用させた絶対アドレスの決定およびタスク間共通データエリアアドレスの決定である。

OLLの場合，オブジェクトとしてタスクを出力するため，ローダ語プログラムのほかにタスクの識別番号や優先レベルなどをパラメータとして与える必要がある。OLLの役割は通常のローダの役割に加えてタスク間共通サブルーチンやタスク間共通データエリアのリンクアドレスの決定と相互参照状況管理テーブルの作成およびメモリのあきエリアの管理を行ない，またローダ語プログラムをPMSの管理下におくために必要な制御管理テーブルの作成を行なっている。

これらローダの役割の中でもタスク間共通サブルーチンやタスク間共通エリアのリンクアドレスの決定は，プログラミング作業の労力軽減に大きな効果をあげている。すなわち，コーデング作業時に，タスク間共通エリアやタスク間共通サブルーチンのアドレスはいっさい意識せず，その名前ですべての参照，更新リンクができるからである。

また，タスク間共通エリアやタスク間共通サブルーチンの相互参照状況管理テーブルの作成により，組合せ試験中やオンライン運転中にバグが発生したときの対策，あるいは内容修正したときの確認範囲の把(は)握に非常に有効である。

さらに，メモリのあきエリアの管理を行なっているので，タスク自身やタスク間共通エリア，タスク間共通サブルーチンのアドレス割付けをユーザーは考える必要はない。この結果，アドレスの二重割付けといった誤りはいっさい発生せず，一方システム取りまとめ者は従来かなりの仕事量となっていたメモリマップ管理作業から開放されることになる。

4　ライブラリシステム[6]

ライブラリシステムはジョブ間で共通に使うサブルーチンをライブラリとして登録する機能で一度登録すると以降の呼び出しは，CALL命令の中でその名前を指定するだけでよい。

このライブラリサブルーチンはすべてバインダ語の形態で記憶されており，メインプログラムとの結合はバインダによって行なわれる。

5　ジョブ制御システム

いくつかのジョブをアセンブル，コンパイル，バインド，あるいは実行させるとき，一つ一つのジョブごとにオペレー

タが介在することなく，連続的に処理できるようにするのがジョブ制御システムである。

ジョブ制御システムはさらに，各ジョブの中で使われる各種の言語処理プログラムが円滑に処理されるように橋渡しも行なっている。**図4**はジョブ制御システムを用いたときのプログラムカード配列例を示すものである。また，このシステムはハードウェアやソフトウェアが異常を起こしたときの異常処理やセンター運営に必要な装置の切換処置などをオペレータに要求したりあるいはオペレータからの各種オペレーションに答えたりして，システムの運営が円滑に行なえるようになっている。

6　デバッギングシステム(7)

いわゆるデバッグといわれるものには，単体のプログラムをジョブとしてNPMSのもとでデバッグする場合と一応単体としてデバッグの済んだものをPMSのもとで実際のハードウェアと組み合わせてデバッグする場合とがある。

後者の場合は，デバッグというよりもハードウェアとソフトウェアのインタフェースを合わせる作業が主体となりチューニングデバッグともいわれる。

以下，HIDICシリーズで提供しているおのおののデバッグサポートプログラムについて述べる。

6.1　オフラインデバッグサポートプログラム

オフラインデバッグを能率よく進めるための機能として下記がある。

(1)　メモリダンプ機能

実行途中や実行結果の内容を，アセンブラに対してはアドレスレベルで，PCLに対しては変数，配列名レベルで印字する。

(2)　トレース機能

プログラムの実行過程の流れをアセンブラに対してはレジスタレベルで，PCLに対しては変数，配列，文番号レベルでトレースする。

(3)　テストデータの自動セット機能

テストデータやプログラム分岐条件を人手を介することなく，被テストプログラムの中にセットし，指定した回数だけ連続的に処理する。

(4)　タスク間リンケージの模擬機能

タスク間リンケージを模擬するため，NPMSのもとでジョブ間共通エリアを確保できる。

このエリアには必要に応じて初期値が設定でき，この初期値を用いて実行した結果のうち，他のジョブに渡すべきデータについては再びジョブ間共通エリアに格納され，次のジョブによって使用される。

この一連のリンケージ確認により，等価的にタスク間リンケージが確認される。

図5はPMSにおけるタスクの動きとそのリンケージを，**図6**はPMSもとでのタスクの動きとリンケージを模擬したジョブの動きとリンケージを示すものである。

(5)　入出力シミュレータ

プロセス入出力装置や，タイプライタ，CRT，トークンC/Rなどのオンライン用入出力装置の処理をソフトウェア的にシミュレートする。

6.2　チューニングデバッグサポートプログラム

チューニングデバッグ時には，プログラムは計算機に組み込んだ形で行なうため，オフラインデバッグ時に比べて，そのデバッグ形態はかなり異なっている。

オフラインデバッグ時には入力指示は主としてカードリーダであるが，チューニングデバッグ時には，入力指示は主としてキーボード入力による。

出力装置も特に印字量の多いものを除いては，コンソール出力装置が主である。

このチューニングデバッグを能率よく進めるために用意されているMCSの機能として下記がある。

(1)　メモリ内容印字修正機能

メモリの内容を印字し，必要に応じて修正する機能である。入力方法としては，絶対番地指定のほか，タスク名，チェーン名を指示したあと，相対番地指定によって行なう方法もある(ここでチェーンとはタスク内のオーバレイプログラムをいう)。

(2)　プログラム修正機能

プログラムの内容をアセンブラ形式で修正する機能である。

(3)　スナップショット機能

プログラムの実行途中の内容を印字したり，実行途中で止めたりする機能で，キーボードから，タスク内の相対番地指定で指定できる。

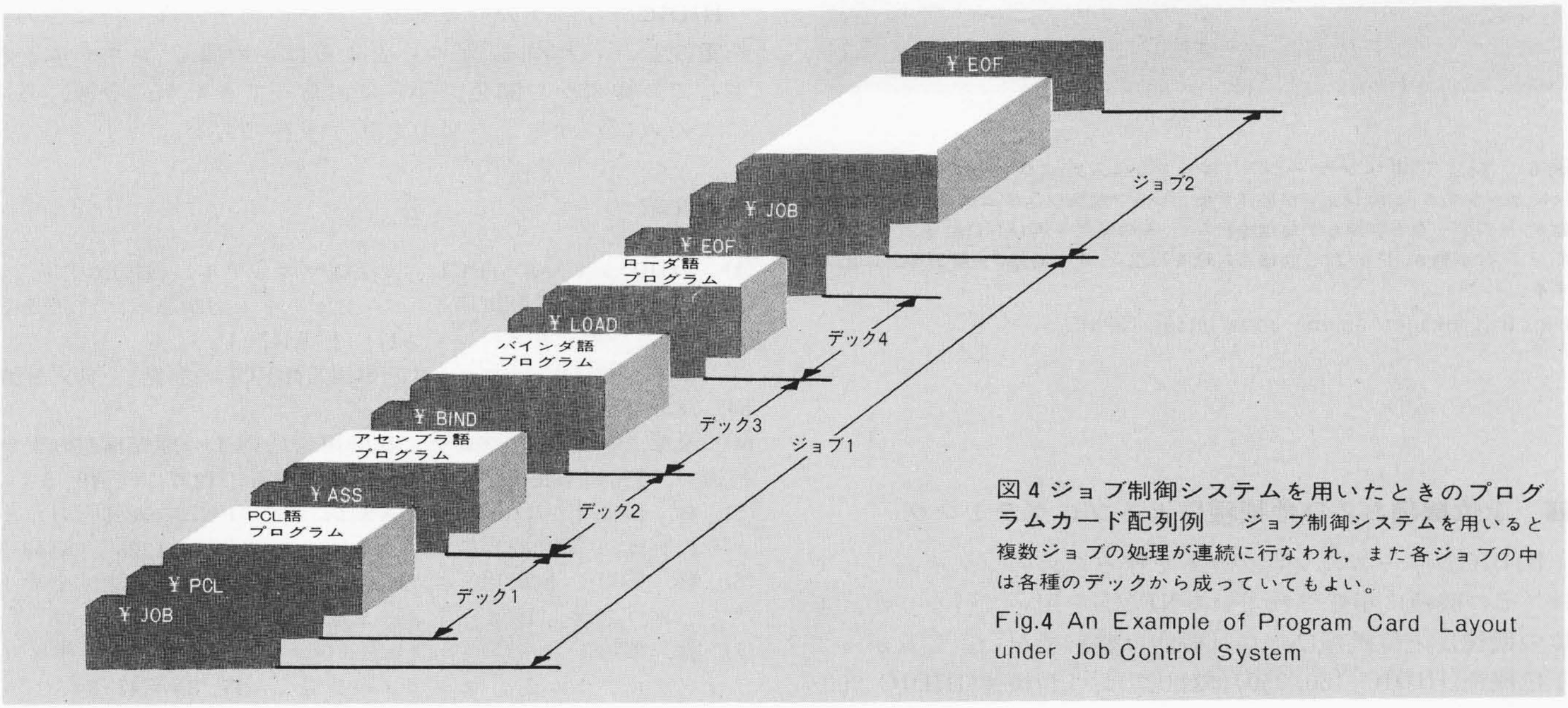

図4 ジョブ制御システムを用いたときのプログラムカード配列例　ジョブ制御システムを用いると複数ジョブの処理が連続に行なわれ，また各ジョブの中は各種のデックから成っていてもよい。

Fig.4 An Example of Program Card Layout under Job Control System

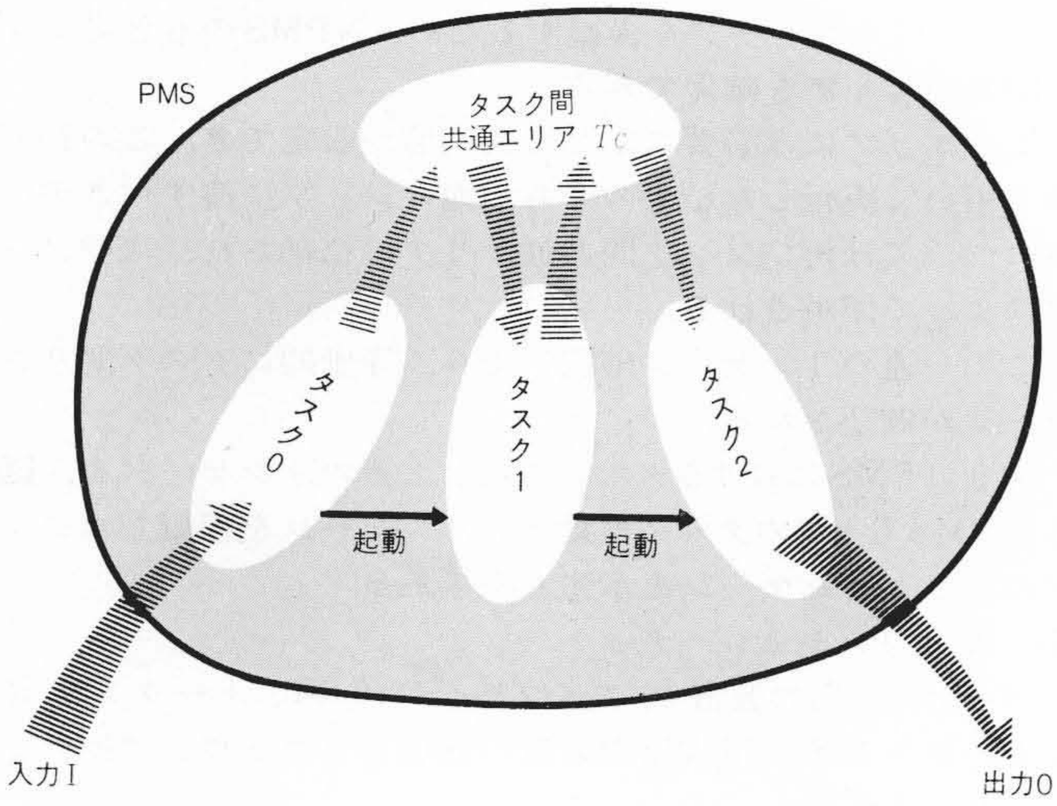

図5　タスク間リンケージ　タスク0は計算機外から入力Iを取り込み，その結果をTcにセットしてタスク1を起動する。タスク1はこのTcのデータを参照して処理を行ない，結果を再びTcにセットしタスク2を起動する。タスク2はタスク1と同様に処理を行ない，その結果Oを計算機外に出力する。

Fig. 5 Linkages among Tasks under PMS

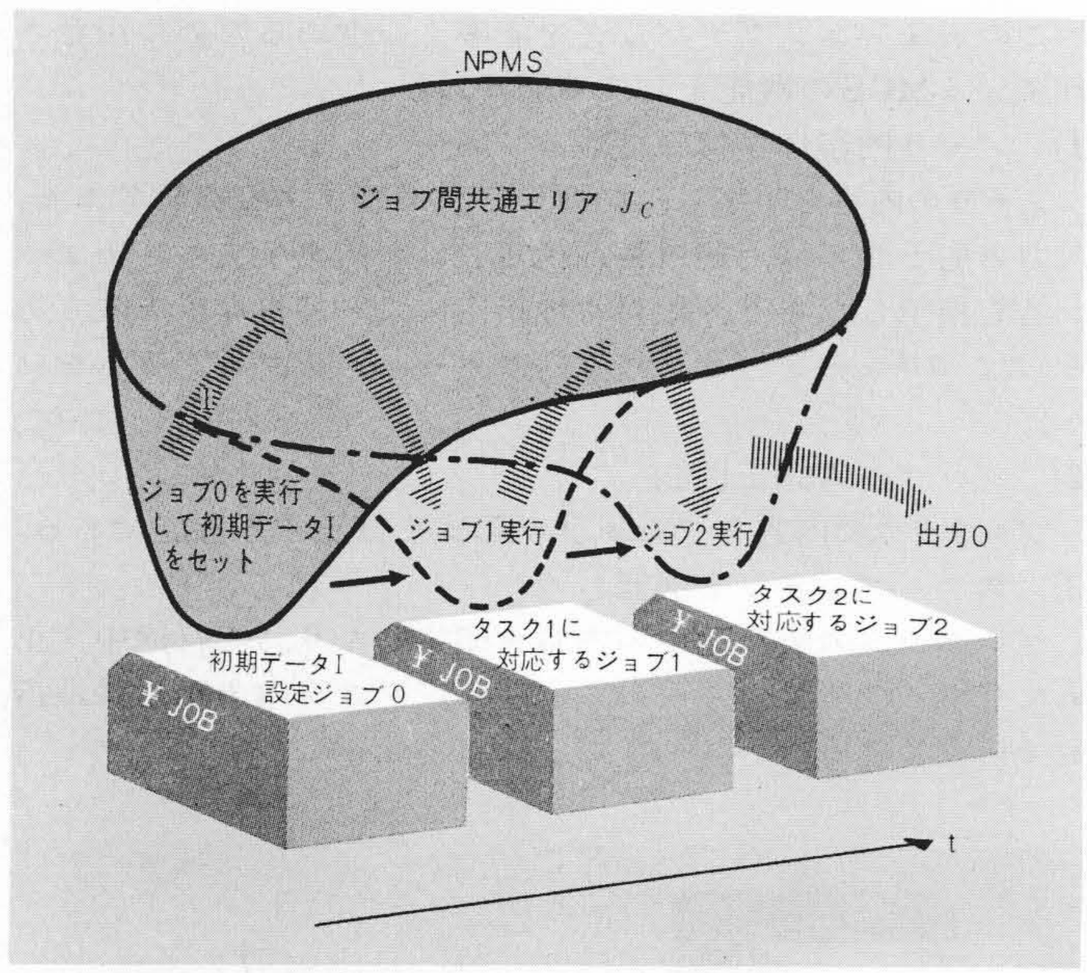

図6　ジョブ間リンケージ　ジョブ0は入力Iに代わって初期データをJcにセットする。このジョブが終了するとジョブ制御システムによりジョブ1が動き，Jcのデータを参照して処理を行ない，その結果を再びJcに格納する。続いてジョブ2が動き，ジョブ1と同様に処理を行ない，その結果Oを計算機外に出力する。

Fig. 6 Linkages among Jobs under NPMS

7　上位機種および他機種によるプログラミング

HIDICシリーズのプログラムを作るためのシステムとしては，その機種に用意されているNPMSを用いて行なうが，下位の機種は上位機種ほどには機能は豊富でない。したがって下位機種(HIDIC 150/350)に対しては，上位機種(HIDIC 500/700, HITAC 7250)でデバッグができ，ローダ語出力もできるようになっている。また，他機種を用いてHIDICシリーズ用のプログラムも作成できるようになっており，使用可能機種として，HITAC 8300up, HITAC 8350up, IBM360/370がある。ただし，他機種を用いる場合に使用できる言語はアセンブラである。**図7**は，これらの関係を示すものである。

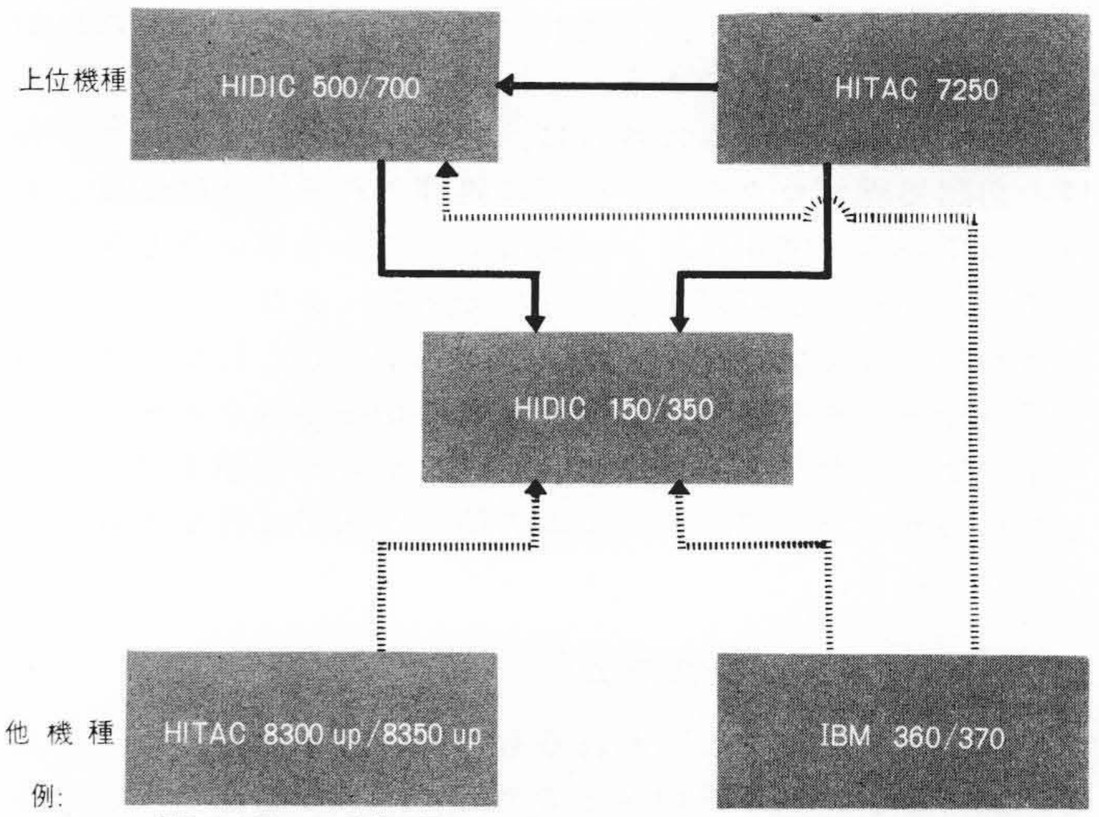

図7　上位機種，他機種によるプログラミング　HIDIC 150/350は上位機種，他機種によってプログラミングでき，上位機種の場合には言語としてPCLを使うことができる。

Fig. 7 Programming Using Upper Grade Machines and/or Different Machines

8　結　　言

以上，HIDICシリーズをささえているプログラミングシステムについてその概要を述べた。このように制御用計算機のソフトウェアを作るにあたっては，非常に多くのサポートプログラムが必要になる。

このプログラミングシステムの全体的な体系は，一般の計算機システムのそれと似ているが，細かい部分では制御用計算機，あるいはオンラインシステム特有の部分があり，この細かい部分の機能の良否によってプログラミング労力は大きく左右される。

HIDICシリーズのプログラミングシステムは，特にこの点を重視し，いわゆる「かゆいところに手の届く」システムをめざして，従来から開発，保守にあたってきたが，今後，さらにこの点にいっそうの努力を払う所存である。

参考文献

(1)　HIDICシリーズ　PCL　文法編マニュアル　(日立製作所)
(2)　桜川，桑原：「制御用ミニコンピュータ入力講座8，言語処理システム」　オートメーション17，1(昭47-1)
(3)　桑原，林ほか：「プロセス制御用言語PCLの開発」　日立評論54，765　(昭47-9)
(4)　桑原，林ほか：「プロセス制御用言語PCLの開発構想および特徴」　昭和48年度信学会全国大会論文集6，1227　(昭48-3)
(5)　林，新納ほか：　「プロセス制御用言語PCLの表現能力およびその評価」　昭和48年度信学会全国大会論文集6, 1228　(昭48-3)
(6)　林，桜川：「制御用ミニコンピュータ入門講座9，ユーティリティ・プログラム(1)」オートメーション17，2　(昭47-2)
(7)　林，桜川：「制御用ミニコンピュータ入門講座10，ユーティリティ・プログラム(2)」，オートメーション　17, 3(昭47-3)