U.D.C. 800. 92:681. 323. 06::62-52

プロセス制御用言語PCLの開発

Development of Process Control Language —PCL

桑原　洋*　*Hiroshi Kuwahara*
林　利弘*　*Toshihiro Hayashi*
福岡和彦**　*Kazuhiko Fukuoka*
坂東忠秋***　*Tadaaki Bandou*

A higher procedural language for use with the control computer—HIDIC 500—has been developed at Hitachi and named Process Control Language—PCL.

The PCL has been developed based on a study on the functions required for process control, and it adopts not only a number of essential functions such as structural description of data, bit handling, decision table processing, virtual memory like data processing, Operating System—Task communication processing, but also many powerful debugging facilities and optimizing method to make object codes shorter.

Consequently, as compared with the conventional programming using assembler languages this language has brought about a remarkable improvement in productivity, visibility, uniformity and maintenability of the software with little sacrifice of memory capacity and running speed of the object codes.

1 緒　　言

プラント制御や工業における省力化への電子計算機の適用が本格化するにつれ，適用業務ごとに作成するプログラムの量は莫大(ばくだい)なものとなってきており，制御用の分野においても，いわゆる,「ソフトウエアの危機」に直面している。

制御用プログラムに対する要求は，

(1) 比較的小規模な計算機が多く，主メモリの制限が大きいので，プログラムはコンパクトにしたい。

(2) 大部分のプログラムは作りつけとなるため，実行能率のよいプログラムを作りたい。

(3) 論理判断処理のプログラムが多いので，その処理を簡単，正確に記述したい。

(4) マルチプログラミング機能を基本としたオンラインリアルタイム性の問題を簡単，正確に記述したい。

などであり，従来のコンパイラではこれらの要求を満たすことができないため，これまで主としてアセンブラ言語が使われてきた。「ソフトウエアの危機」を打開するためには，このアセンブラ言語を制御用として必要かつ十分な機能と性能を有する高級言語に置き換え，ソフトウエアの生産性，可視性，均一性および保守性を上げることが良策である。このたび，これらの要求を満たす高級言語，ＰＣＬ（Process Control Language）を開発したので，以下にその機能，特長などについて述べる。

2 開発の方針

2.1 PCLの位置づけ

ＰＣＬは，**図1**に示す日立制御用計算機HIDIC 500言語処理システムの頂点に立つものである。

HIDIC 500言語処理システムのうちプログラミング言語はＰＣＬ，フォートラン，アセンブラの三つである。

これらプログラミング言語のうち，フォートランは制御用の分野においては非常にまれな複素数を含む特殊計算プログラムの作成に，またアセンブラは主メモリに常駐して複数のタスクから共通に用いられる再入可能なサブルーチンの作成に用いられ，これらの特殊なものを除いた制御用プログラムのほとんどはＰＣＬによって作成することをねらっている。

なお，これら三つのプログラミング言語の使用量の比を示

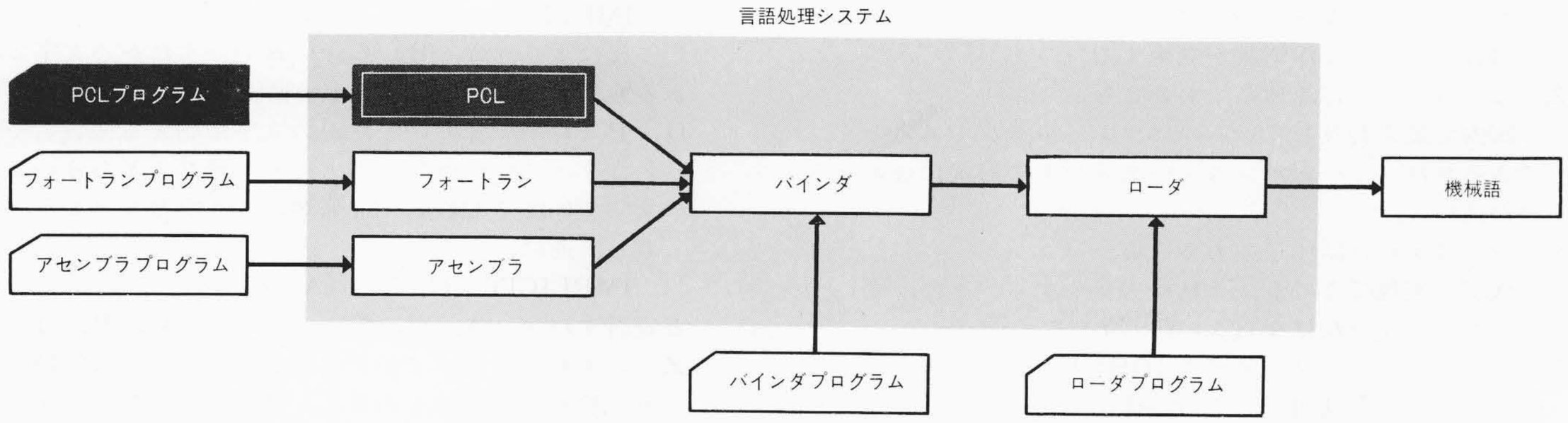

図 I　HIDIC 500言語処理システム　PCL，フォートラン，アセンブラはプログラムを記述するための言語である。バインダはこれらの種類の異なる言語で記述されたプログラムを結合してより大きな一つのローダプログラムにするものである。また，ローダはこのローダプログラムを計算機が実行できる形（機械語）にして計算機に組み込むものである。

Fig. I　HIDIC 500　Language Processing System

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

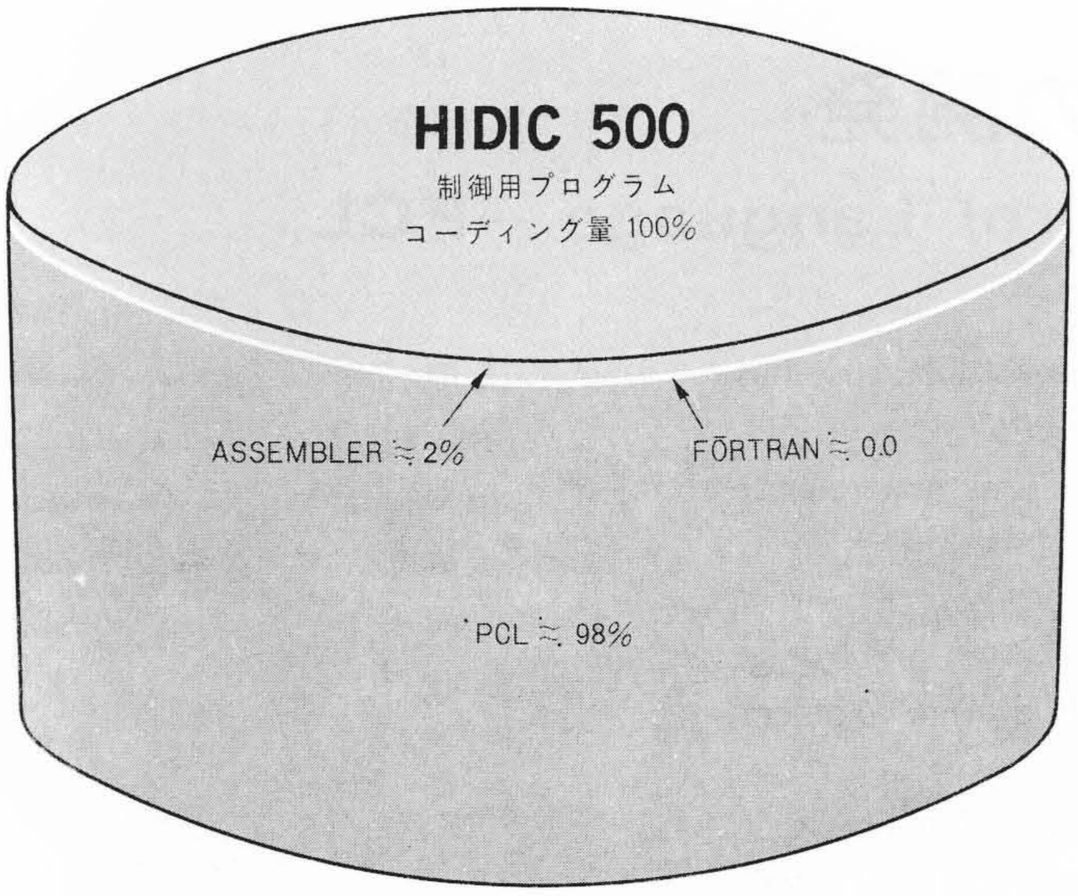

図2　プログラミング言語の使用比率　　制御用プログラムの約98%はPCLで作成し，残りの約2%のうち複素数を含む特殊プログラムはフォートラン，主メモリに常駐して複数のタスクから，共通に用いられる再入可能なサブルーチンはアセンブラで作成する。

Fig.2　Usage Rate of Programming Languages

したのが**図2**である。

2.2　言語仕様

ＰＣＬの開発にあたって，その言語仕様は下記の方針に従って決定された。

(1)　アセンブラ言語を混用しなくても，制御用として十分使用に耐えうる言語機能を有すること。

従来の制御用コンパイラといわれて発表されているものの中には，アセンブラ言語との混用を許しているものがいくつか見られるが，これはコンパイラになじんでいるものには苦痛であり，一方，アセンブラになじんでいるものにはコンパイラを軽視する風潮を与え，いずれにしても高級言語のもつ特徴が十分生かせなくなる。

(2)　アセンブラ言語で書かれた既開発プログラムが有効利用できる言語体系であること。

(1)にかかわらず，既製のアセンブラコーディングプログラムが有効利用できることは重要である。

一般に，アセンブラ言語によって作られたサブルーチンのリンケージ法は，コンパイラ言語で作られたものと異なるため，この差異を既製サブルーチンの改造によって行なう場合には大きな作業量が発生する。

(3)　なじみやすい言語体系であること。

国内で最も普及しているのはフォートランであるので，できるだけフォートランステートメントの形式に合わせるのがよい。

(4)　使いやすい言語体系であること。

機能が実現できても，それが書きやすく，また，使いやすいものでなければならない。

(5)　オペレーティングシステム（ＯＳ）のマルチプログラミング機能を有効に活用できる言語体系であること。

個々のプログラムの効率がいくら良くても，システム全体として見たときの効率が悪くては意味がない。その意味からいって，ＯＳの機能，特にマルチプログラミング機能を有効に生かせる言語体系でなければならない。

(6)　デバッグのことを十分考慮した言語システムであること。

プログラム開発作業の半分以上はデバッグといわれているが，このときに大きな効果を発揮する言語システムでなければならない。

(7)　入出力機能を特に強化した言語体系であること。

制御用計算機システムの場合，プロセス入出力装置やカラーＣＲＴなどの特殊入出力装置が数多く使われ，これらの記述が能率よくできなければ，プロセス制御用言語として満足のいくものとはいえない。

2.3　言語性能

言語性能に対する評価要素としては，

(1)　コンパイルに要する時間　(2)　オブジェクトの大きさ

(3)　オブジェクトの実行時間

がある。ＰＣＬでは，これら三つの要素に対する優先度を，(2)，(3)＞(1)とし，(2)と(3)の優先度については，ステートメントごとに検討して決めることにした。

3　PCLの特長

ＰＣＬは，制御用プログラミングの特徴を分析し，言語として必要な機能と性能の目標値を設定し，これを実現したものである。

したがって，個々のステートメントは任意の形式を取り得たが，前記の方針に沿って，なじみやすさを考慮し，できるだけフォートランと同じ記述形式にした。しかしシンタックスは当初の目標を実現するために，フォートランをかなり拡張したものとなっている。

以下にＰＣＬのおもな特長を，仕様上の特長と性能上の特長に分けて述べる。

3.1　仕様上の特長

(1)　整数型データを使いやすくした宣言文

制御用プログラムにおいては，積算データ処理に見られるように，整数演算を行なわなければ積算誤差や丸め誤差が発生するものが多い。これに対して，従来からあるフォートランは暗黙の型宣言で整数型となるものはＩ，Ｊ，Ｋ，Ｌ，Ｍ，Ｎで始まる英数字だけで，それ以外の文字で始まるものについては個別に整数型であることを宣言せねばならず，大きな手間が発生していた。ＰＣＬではIMPLICIT文を設けることにより，暗黙の型宣言の規約を任意に拡張でき，Ｉ，Ｊ，Ｋ，Ｌ，Ｍ，Ｎ以外の文字で始まる英数字に対しても個別に宣言することなく整数型であることが指定できる。

（例）　IMPLICIT　INTEGER　（A－N）

IMPLICIT　INTEGER＊2　（O－X）

これにより，Ａ，Ｂ，Ｃ……Ｎのいずれかの文字で始まる変数名や配列名はすべて単精度整数型とみなされ，Ｏ，Ｐ，Ｑ……Ｘのいずれかの文字で始まる変数名や配列名はすべて倍精度整数型となり，残りのＹあるいはＺの文字で始まる変数名や配列名のみが実数型となる。

もし，逆に，

IMPLICIT　REAL　（A－K）

と宣言すれば，Ａ，Ｂ，Ｃ……ＫおよびＯ，Ｐ，Ｑ……Ｚのいずれかの文字で始まる変数名や配列名は実数型となり，整数型となるものはＬ，ＭあるいはＮのいずれかの文字で始まる変数名や配列名のみとなる。

(2)　構造指定による階層形式データの記述

各種のテーブル類は，通常いくつかのデータ要素から成っており，これらのデータ要素は１～２語あるいは数ビットであることがある。これらの各データ要素やその集合体に名前をつけ必要に応じてデータ要素名や集合名で参照できるようにしたのが，この構造指定による階層形式データ

の記述で，PL/Iと同様に構造体と呼んでいる。

（例）　1I　（2），
　　2J　（2），
　　3A：BIT　（8），
　　3X：BIT　（8），
　　2K　（2），
　　2L，
　　3M　（2），
　　3N，

上記構造体のメモリ上での割付け状態を示したのが**図3**である。

(3)　豊富なビット処理機能

接点，スイッチなどに代表される1ビットデータの処理，シーケンス制御に代表されるデシジョンテーブルの処理，さらに，プログラムの内部処理時によく現われる数ビット長データの処理などが通常のデータと同じレベルで，体系的に記述できる。

（例）　BIT　DIMENSION　A　（20）
　　BIT　DATA　A/10110101，12＊1/

この例ではBIT　DIMENSION文により，A(1)からA(20)までビット単位に配列要素名が与えられ，おのおのにはBIT　DATA文により，A(1)には1 A(2)には0……A(8)には1，A(9)からA(20)まではすべて1なる初期値が与えられる。

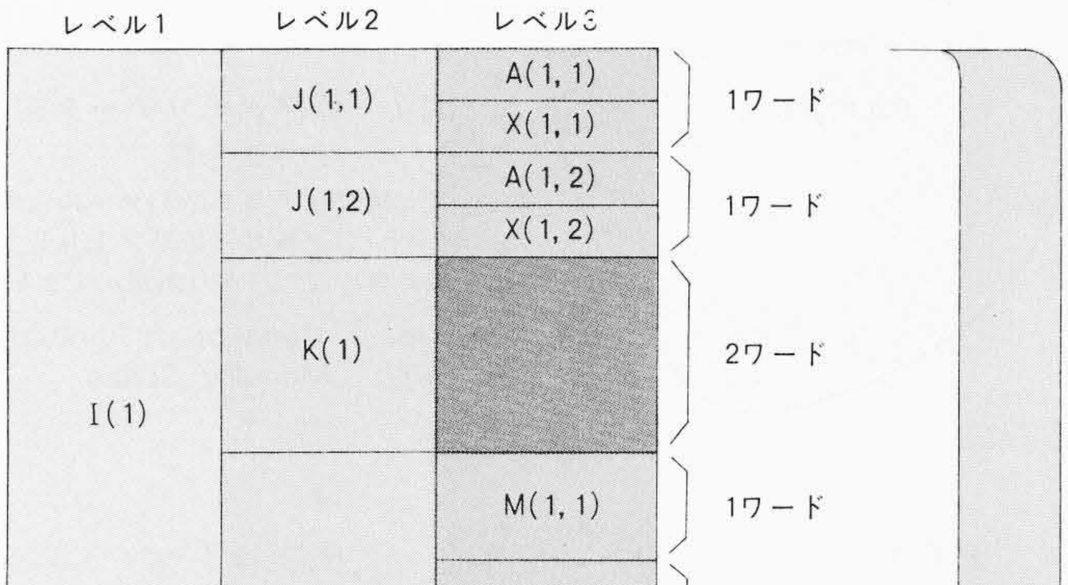

図3　構造体のメモリ上での割付け状態　　Iは I(1)と I(2)の2組から成り（レベル1），各Iはさらに2組のJとKおよび1組のLから成り（レベル2），Jはさらにおのおのが A，Xという8ビットデータ，Kはさらに2組のMと1組のNから成っている（レベル3）ことを示し，全体で14ワードのメモリを占める。

Fig.3　Memory Layout of Data Structure

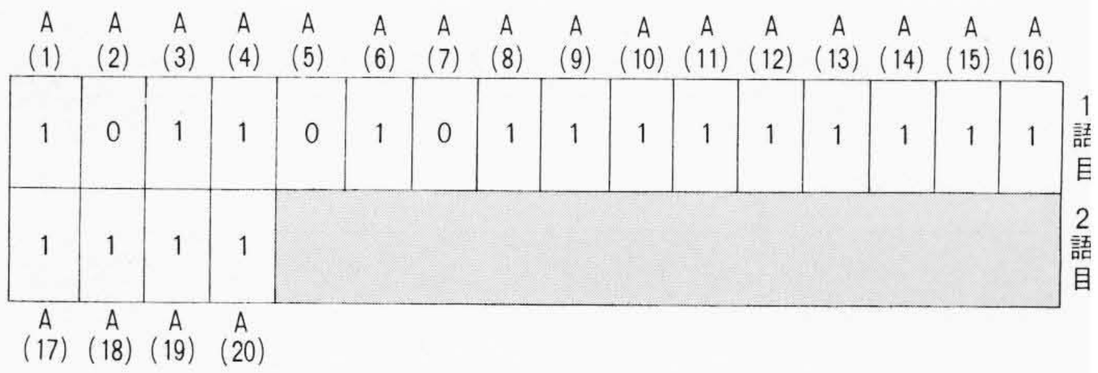

図4　Bit　DimensionとBit Data　　AはA(1)からA(20)までの計20ビットのエリアを持ち，A(1)は1，A(2)は0……A(20)は1なる初期値を有していることを示す。なおA(21)からA(32)に対応する部分はエリアは確保されるが定義されていないため，使えないことを示す。

Fig.4　Bit Dimension and Bit Data

この様子を図示したのが**図4**である。

このようなBIT型配列を宣言するものとしては，このほかにBIT　COMMON文，BIT　GLOBAL文があり，いずれも最高3次元まで規定できる。

（例）　デシジョンテーブル処理

図5に示す20入力，5ステップのデシジョンテーブルをコーディングすれば**図6**のようになる。

（例）　ビットストリング式

A，X，Nを**図3**に示す構造体のA，X，Nとすれば，

N(1)＝A(1，1)＆X(1，1)＋N(2)

はA(1，1)とX(1，1)に対してビット単位のANDをとった結果を正整数とみなし，これにN(2)を加えた結果をN(1)に代入することを示す。

ビットストリング式は一般に下記のように定義されている。

I：1100110011001100
J：　　11110000

のとき，

┐I：0011001100110011
I＆J：0000000011000000
I｜J：1100110011111100
I％J：1100110000111100

(4)　タスク間にまたがるデータに対する宣言文

制御用計算機では通常複数個のプログラムがマルチタスキングされて動くが，このとき，タスク相互間でデータをやり取りするためのエリアが必要となる。

このエリアは，通常，主メモリ上の常駐エリアや補助メモリ上に確保されるが，従来はそのエリアを参照するのに絶対番地を用いたり（アセンブラの場合），もしくは不可能であったり（フォートランの場合）した。これに対して，PCLでは必要なエリアを変数名や配列名として宣言するだけでよく，主メモリ上や補助メモリ上への割付けはローダが自動的に行なう（**図1**）。

（例）　GLOBAL文
　　BIT　GLOBAL文。

これらは主メモリの常駐部にデータエリアを確保する。

　　BULK文

これは補助メモリ上にデータエリアを確保する。

いずれも，最高3次元の配列まで規定できる。

(5)　仮想メモリの概念をもつデータ

プログラミングの専門家でなく，システムエンジニアやプラントエンジニアがプログラムを作成する際，補助メモリ（ドラムメモリ）上のデータに対しても特に意識することなく，主メモリ（コアメモリ）上のデータと同様に各種

テーブル名A	入力 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	アクション	
テーブル名C	条件																				条件成立時	条件不成立時
ステップ1	1	1	X	0	1	X	X	X	X	X	0	0	0	0	0	1	1	1	1	1	ACT1を実行してステップ2へ	10秒後ステップ1へ
ステップ2	1	1	1	0	X	1	0	1	0	1	1	X	X	X	X	0	1	1	0	1	ACT2を実行してステップ3へ	10秒後ステップ2へ
ステップ3	1	0	0	1	X	X	1	1	0	X	X	1	0	1	0	0	X	X	X	X	ACT3を実行してステップ4へ	10秒後ステップ3へ
ステップ4	X	0	0	1	0	X	1	X	0	0	X	1	0	1	0	0	0	X	0	0	ACT4を実行してステップ5へ	10秒後ステップ4へ
ステップ5	0	0	0	1	0	X	X	X	0	0	0	X	X	1	0	X	0	X	0	X	ACT5を実行してSTOP	10秒後ステップ5へ

図5　デシジョンテーブルの例　入力1～入力20には各種の情報が別タスクにより0,1の2値情報に変換した形で一定時間ごとに格納されるものとし，条件欄の1は対応する入力が1であること，0は対応する入力が0であること，Xは対応する入力が1でも0でもよいことを規定する。このとき，上記デシジョンテーブルは入力A全体と条件Cの各ステップをビットごとにチェックし，すべてのビットに対して条件が成立したか否かに従って，アクション欄に示した処理を行なうことを示す。

Fig.5 Example of Decision Table

```
PAGE 0001 HIDIC 500  COMPILE  LIST
  EFN                          PCL SOURCE STATEMENT
  C  DECISION TABLE PROCESSING TASK : DECTK
       IMPLICIT INTEGER  (A-Z)
       BIT DIMENSION     C(5,20)
       DECISION DATA     C(1) /11X01XXXXX0000011111/,
       1                 C(2) /1110X101011XXXX01101/,
       2                 C(3) /1001XX110XX10100XXXX/,
       3                 C(4) /X0010X1X00X101000X00/,
       4                 C(5) /00010XXX000XX10X0X0X/
       BIT GLOBAL  A(20)
       VALUE    T1,DECTK
  C  N1=TIMER NO; TIMER UNIT=0.1SEC,YUENI  10SEC=100
  C
       DO  50  I=1,5
  10   DECISION IF (C(I),EQ,A) GO TO (1,2,3,4,5),I
       TIMER   T1,100,DECTK
       WAIT
       GO TO 10
   1   CALL ACT1(&50)
   2   CALL ACT2(&50)
   3   CALL ACT3(&50)
   4   CALL ACT4(&50)
   5   CALL ACT5(&60)
  50   CONTINUE
  60   STOP
  C  CALL ACT(&50) WA SUBROUTINE NO JIKKO NO ATO
  C                        50 E JUMP SURUKOTO O SIMES.
       END
```

図6　デシジョンテーブル処理プログラムのコーディング・リスト例　図5のデシジョンテーブルをPCLステートメントを用いてコーディングした例である。入力AはBIT GLOBAL A(20)によりコア常駐エリアに，条件のステップ1～5はBIT DIMENSION C(5,20)とDECISION DATA C(1)～C(5)によりこのタスクの中にエリアと初期データがとられることを示し，文の番号(EFN)10で示されるDECISION IFステートメントによりこれらの入力と条件のチェックが行なわれる。

Fig.6 Example of Coding for Decision Table Processing

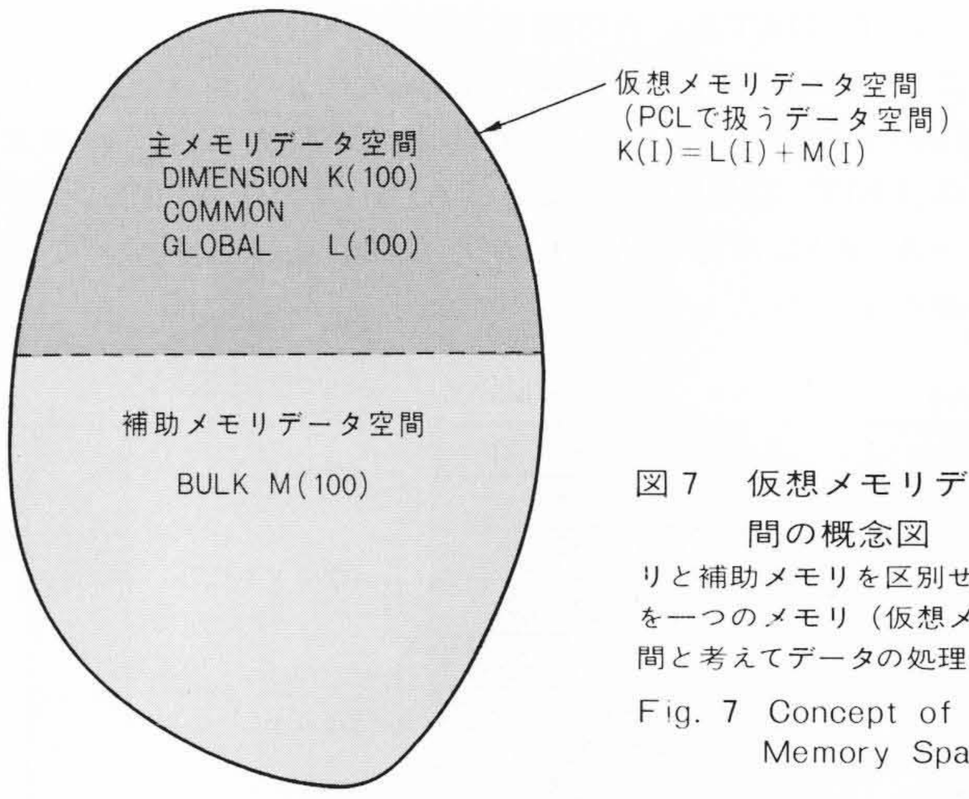

図7　仮想メモリデータ空間の概念図　主メモリと補助メモリを区別せずに全体を一つのメモリ（仮想メモリ）空間と考えてデータの処理ができる。

Fig. 7 Concept of Virtual Memory Space

ステートメントの中で直接扱える。**図7**はPCLで採用した仮想メモリの概念を，**図8**はコーディング例を示したものである。

(6)　システム交信文の組込み

一般に，制御用のオンラインリアルタイムシステムにおいては，独立に動いている各タスク間の同期をとるために，他のタスクを止めたり，動かしたりあるいは一時停止させたりし，またタスク間で共通に使われる資源に対しては競合やデッドロックが起こらないようにすることが必要であるが，これらの信号のやりとりはすべてＯＳの制御命令を介して行なわれる。

ＰＣＬにはこれらＯＳの制御命令の機能はすべてシステム交信文として組み込まれている。

（例）　QUEUE文，START文，STOP文，
WAIT文，RESERVE文，FREE文，……。

(7)　豊富で拡張性のある入出力機能

Ｌ/Ｐ，Ｃ/Ｒ，ＰＴＲ，ＰＴＰなどの一般I/Oはもとより，ＡＩ，ＤＩ，ＡＯ，ＤＯなどのプロセス入出力装置やカラーＣＲＴまでも，同じ思想のもとで同じステートメントで記述でき，かつ個々のI/Oの特異性が言語上に現われないように考慮されている。

図9はカラーＣＲＴに文字を表示するプログラムのコーディング例を示したものである。この例において，WRITE文の中の70をカラーＣＲＴディスプレイを示すファイル参照番号とすれば，このプログラムの実行により，Ｋの値（1000）が整数型5けたで赤色表示される。

(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

アセンブラ言語で作られたサブルーチンとコンパイラが作り出したサブルーチンの引数処理方式を比べた場合，前者は引数の値そのものが直接セットされることが多く，後者は引数そのものはセットされず，引数の値が格納されている番地がセットされる。このため，一般にアセンブラ言語のサブルーチンをそのままコンパイラ言語で作ったプロ

HIDIC PCL CODING SHEET

PROGRAM	CODED BY DATE :	DECK ID. [73 74 75] PAGE [76 77] OF ____

COMM 1	STATEMENT NUMBER 2–5	CONT. 6	PCL STATEMENT 7–72	DECK ID. 73–75	PAGE 76–77	SEQ. NO. 78–80
C			VIRTUAL MEMORY SPACE			000
			DIMENSION K(100)			050
			GLOBAL L(100)			100
			BULK M(100)			150
			DO 100 I=1,100			200
	100		K(I)=L(I)+M(I)			250
			STOP			300
			END			350
						400
						450
						500
						550
						600
						650
						700
						750
						800
						850
						900
						950

Hitachi,Ltd.

図8　仮想メモリデータ空間を用いたプログラム例　　コア常駐部にあるデータL(I)と補助メモリ部にあるデータM(I)を加えて，タスク内のエリアK(I)に格納するプログラムがK(I)=L(I)+M(I)と書け，特にデータの存在する場所を式の中で意識する必要はない。

Fig.8 Example of Coding Using Virtual Memory Data Space

```
C   COLOUR CRT DISPLAY
    INTEGER K
    K=1000
    WRITE (70,100) K
100 FORMAT(!RED,I5)
    STOP
    END
```

図9　カラーCRTディスプレイへの表示プログラム例
WRITE文中の70がカラーCRTディスプレイを示すものとすれば，このプログラムの実行によりKの値(1000)が整数型5けたで赤色表示される。

Fig 9 Example of Coding for Colour CRT Display

グラムからコールしてもうまくリンクせず，アセンブラ言語のサブルーチンのほうを変更しなければならないのが通常である。

これに対してPCLはいずれのリンク方式に対しても処理できるため，アセンブラ言語のサブルーチンをコンパイラ語で作ったプログラムの中からコールする際にもなんらの変更作業を伴わない。

(例)　CALL SUB (A, 10)
　　CALL SUB (@A, @10)

引数の前に@のある場合は通常のコンパイラの引数処理とは異なり，アセンブラ言語によるコーディングの際に通常行なわれる引数処理方式をとる。

(9)　豊富なデバッグ機能

ダンプ，トレース機能はもちろん，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コンパイルモード指定機能や使用サブルーチンの切換え機能，さらに配列添字のオーバフローチェック機能などを持ち，コンパイル時や実行時のエラーメッセージは数百に及んでいる。

(例)　DEBUG VALUES文

これは指定された文番号の範囲内で，指定された変数や配列に値が代入されたとき，その変数名や配列要素名と代入された値をL/Pに印字する。

DEBUG FLOW文

これは指定された文番号の範囲内でGOTO文，算術IF文あるいはRETURN文を実行したとき，どこからどこへ制御が移ったかを印字する。

DEBUG SUBSCRIPT文

これは指定された配列において，その配列要素名の添字の値が宣言された値より大きいとき，その旨を印字する。

COMPILE MODE文

この文の使用例を示すと**図10**のようになる。

この例ではWRITE文とFOMAT文は注釈文として処理される。もし**図10**において，COMPILE MODEが%指定であれば，そのコンパイル結果は**図11**に示すようになる。

RENAME文

この文の使用例を下記に示す。

```
RENAME (FETCH, DUMP)
    ⋮
CALL FETCH (A, B, C)
```

この例では，CALL FETCH (A, B, C)は，CALL DUMP (A, B, C)として実行される。

```
C     DEBUG FACILITY (COMPILE MODE)
      COMPILE MODE *
      GLOBAL L(100),M(100),N
*     DEBUG VALUES 1,100
*     DEBUG SUBSCRIPT L,M
*   1 CONTINUE
      DO 100 I=1,N
  100 L(1)=M(I)+N
%     WRITE(6,200) L
% 200 FORMAT(1H ,2015)
      STOP
      END
```

図10　コンパイルモード＊の使用例　　コンパイルモードが＊の指定であるので第1欄が%の文は注釈として扱われる。

Fig.10 Example of Usage of Compile Mode ＊

```
PAGE 0001 HIDIC 500  COMPILE  LIST
   EFN                          PCL SOURCE STATEMENT
   C  DEBUG FACILITY (COMPILE MODE)
         COMPILE MODE %
         GLOBAL L(100),M(100),N
         DO 100 I=1,N
     100 L(1)=M(I)+N
   %     WRITE(6,200) L
   % 200 FORMAT(1H,2015)
         STOP
         END
```

図11　コンパイルモードを%と指定したときのコンパイル結果例
コンパイルモードを%と指定して，図10のプログラムをコンパイルした結果であり，図10中の＊を第1欄に持つ文はすべて無視されている。

Fig.11 Example of Compilation by Compile Mode %

3.2　性能上の特長

(1)　16ビットマシンを生かした効率よいオブジェクト

16ビットマシンは通常，すべての命令に対して，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，PCLでは，できるかぎり短語形式を用いるようにし，その結果，効率よいオブジェクトを生成している。

従来の16ビットマシンに対するコンパイラにおいて，そのオブジェクトがアセンブラ言語によるプログラムと比べて効率が悪いとされていた一つの点はここにあったが，PCLではこの問題を解決することにより，大幅な効率の向上を図っている。

(2)　補助メモリの効率よい利用を目ざしたオブジェクト

変数名や配列名で規定されるエリアは，DATA文によって初期値の設定されたものを除いてすべて実行時に値が格納されるワークエリアであり，プログラムが実行のために補助メモリから主メモリにロールインされたときにそのエリアを確保すればよいが，従来のコンパイラではプログラムを主メモリで動かすことのみを考えていたため，作りだしたオブジェクトをマルチプログラミングの対象として補助メモリ上にたくわえた場合，補助メモリ上にも変数や配列に対応するワークエリアが確保され，補助メモリ容量増大の一因となっていたが，ＰＣＬでは解決されている。

(3)　従属プログラムの共通化

従来のコンパイラオブジェクトでは，従属プログラム（たとえば，実行時のREDA/WRITE処理プログラム，型変換プログラム，各種関数など）は各メインプログラムのあとについてくるが，一般には，二つ以上のメインプログラム（ＰＣＬではタスク）で共通に用いるものに対しては主メモリに常駐化することにより，個々のプログラムの主メモリ上および補助メモリ上での大きさを大幅に削減できる。このため，ＰＣＬではコンパイラが自動的に作り出す従属プログラムはすべて再入可能な形で提供され，容易に主メモリに常駐化させて，二つ以上のタスクで同時に使用できるようになっている。

(4)　オーダーエントリー方式の従属プログラム

従属プログラムは一つの機能に対して一つのプログラムだけを用意するのではなく，いくつかのプログラムを用意している。

たとえば，機能は全く同一であるが，一方は語数は少し大きいが実行時間は短く，他方は語数は短いが実行時間は少し長いという2種類のプログラム（リンク名は同じ）がある。このプログラムの使用ひん度の高いユーザーは前者を，使用ひん度の低いユーザーは後者を採用すればよい。

(5)　入出力処理プログラムのタスク化

従来のコンパイラの入出力処理プログラムは，入出力のフォーマットおよび並び要素の処理のために従属サブルーチンになっているが，これにはＣＰＵと入出力機器の並列処理を行なうマルチプログラミングの概念がはいっていない。

ＰＣＬでは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この点を解決し，あわせて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，低速入出力機器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。

(6)　その他の各種最適化手法の採用

上記(1)～(5)のほかにもステートメントレベルでの各種最適化手法，たとえば，DO　LOOPの最適化，配列添字計算の最適化，算術IF文の最適化，GOTO文の最適化などを採用し，実行時間とメモリスペースの極小化を図っている。

上記(1)～(6)に述べた各種最適化手法の採用により，個々のプログラムのメインルーチンについては，中堅プログラマの作成したアセンブラコーディングプログラムに対して，1.0～1.1倍，従属プログラム，ＯＳも含めた一つの計算機システムについては，すべてアセンブラで作成したものに対して，メモリサイズで1.15～1.25倍であることが確認されている。

4　結　　言

HIDIC 500用として開発されたプロセス制御用コンパイラＰＣＬの開発方針と言語の仕様上および性能上の特長について述べた。

従来，プロセス制御用もしくはオンライン用のプログラムといえばアセンブラというのが常識であったが，ＰＣＬの開発を契機にプロセス制御やオンラインといえども，高級手続き向け言語の使用が一般化し，少しでも「ソフトウエアの危機」を打開できれば幸いである。

ＰＣＬは，現在HIDIC 500用となっているが，他のHIDICシリーズ用にも使えるように開発中である。

終わりに臨み，本開発にご協力いただいた関係各位に深く謝意を表する。