D 1. システム記述用言語PLDの使用経験と問題点

伊藤哲史（日本情報処理開発センター）

はじめに

PL／Iがシステム記述用にも有効な言語であると宣伝されてすでに久しいが，オペレーティングシステムやランゲージプロセッサの何パーセントが，PL／Iで書けるかとか，その結果のオブジェクトの効率がどうであるかという具体的な事になると当然ケースバイケースであって実際やってみないと適当な評価が出来ない．

たまたま当センターで会話型PL／I（Conversational PL／I：略してCPLと呼ぶ）を作成することとなり，この際従来のアセンブラ言語一辺倒をやめて，何かシステム記述用言語を使ってどのような問題点がでてくるかを実験してみることとなった．

この場合，今までに何種類かのランゲージプロセッサを開発した経験から考えて，次のような機能を持っている言語が望ましいということになった．

① 実行時間が速いこと．

② ビット，キャラクターの扱いが簡単に出来ること．

③ CPLが会話型のプロセッサであるためにオブジェクトプログラムはリエントラントであることが望ましい．

④ CPLに限らず他のシステムプログラム用にも利用できること．

汎用言語を使ってランゲージプロセッサを作るということは，注意深く設計，コーディングすれば他の機種にも比較的簡単にコンバートできるという点で非常にすぐれている．

この点と上記4点とをにらみあわせてみると，PL／Iがかなり有望になるのであるが，CPLの開発にかかった時点ではまだFACOM 230−60ではバッチ用PL／Iは使用できる状態ではなかった．又ALGOL，COBOL，FORTRANでは②，③の条件は満たされない．一つの方法として，CPLの基本的な部分だけをアセンブラで作りあげ，CPLによって順次システムを拡大してゆくということも考えられるが，この方法はCPLが会話型プロセッサであるために実行時間が遅くなること，及びCPLがリエントラントなオブジェクトを作り出さなければならないことから見送った方がよいとの結論に達した．そこで最後にPL／Iの機能の中でランゲージプロセッサを作り上げるために最小限度必要なものだけをとりだしてPL／Iのサブセットを作り上げることの検討をした．

この結果PL／Iサブセットであると，MONITOR－Vが持っている機能をフルに生かそうとする問題点が残ることおよび前述の①から④までのすべてを満足するPL／Iサブセットを作るだけの時間的な余裕がないとの2つの理由からランゲージプロセッサを作るのに最小限度必要な機能を持った言語を作ることになり，これをPLDと名づけた．

1. PLDの言語仕様

システムプログラムの作成のみを目的とするならば，その機能はごく簡単なものでよいのではないだろうか．例えば，数式や入出力の機能には相当な制限を加えても支障はないと思われる．

CPL作成に際して特に必要と思われた機能は次のような点である．

① 比較

② キャラクタの取扱い

③ ビットの取扱い

　○ビットのオン・オフ及びその検査機能

　○シフトの機能

　○論理演算

④ 異なる変数名及び属性で同一データを参照できること

⑤ 変数が割り当てられたアドレスを求める機能

⑥ 間接アドレス参照

⑦ テーブルの索引

PLDではこれ等の機能のうち④以外の機能は全て満足するように作成されている．④の機能については現時点では一部属性のチェックを省略することによって同一データを異った属性で呼びだせる様にしている．

以下に各ステートメントの簡単な説明を行なう．

（1）PROCEDURE文

PROCEDURE文は，以下の文がPLDで書いたプログラム単位であることを宣言すると同時にその入口名を定義する．

一般形

ラベル：PROCEDURE〔OPTIONS(｛MAIN / A，数値定数 / B，　〃｝)〕；

labelはこのPROCEDUREの入口を示している．

OPTIONS(MAIN)は書かれていれば，これがMain Programであり，書かれていなければSubprogramとみなされる．

PLDで書かれたProgramの中に表われる変数は，A群とB群の2つのグループにわかれてAreaが確保されそれぞれのグループはプログラム単位となり，Main Program の実行の最初の段階でLOADされる．

＊1. 'PROCEDURE' と書く代りに略称として 'PROC' と書いてもよい．

＊2. PROCEDUREは入れ子にすることは出来ない．

＊3. OPTIONSのカッコの中に(A，数値定数)あるいは(B，数値定数)と書いた場合は，このPROCEDUREはAあるいはB群の中にとられる．数値定数はこのPROCEDURE

が占めるWORD数を指定すればよいが，実際のWORD数より大きいか等しくないと満足な結果は得られない．

（2）DECLARE文

DECLARE文は名前に対しそれが変数名，配列名であることを宣言し，同時にデータの属性を定義する．

一般形

DECLARE〔BLOCK($\left\{\begin{matrix}A\\B\end{matrix}\right\}$)〕指定，指定，‥‥，‥‥，指定；

BLOCK($\left\{\begin{matrix}A\\B\end{matrix}\right\}$) は書いてあればここに宣言された名前がすべてA群かB群に属することを示している．省略されればA群とみなされる．

指定：$\left\{\begin{matrix}\text{変数名}\\\text{配列名（数値定数）〔VARY BASED n〕}\end{matrix}\right\}$属性，属性，‥‥

nは1，2，4，5，6

属性は次の通りである．

INTERNAL or EXTERNAL,

BIT or FIXED or CHARACTER〔(数値定数)〕 or ADDRESS

EVEN

INITIAL（Initial定数指定）

属性の指定は必要なだけ書くことが出来るが，その間に矛盾があってはならない．

INTERNAL，EXTERNAL

INTERNALはこのPROCEDURE内だけで参照出来る名前であり，EXTERNAL は外部のPROCEDUREで同じ名前がEXTERNALの宣言がされていれば同一の番地を指している．なおEXTERNAL宣言はMain Routineで必ずDECLAREされねばならない．

BIT，FIXED，CHARACTER，ADDRESS

これらは，それぞれ BIT 変数，数値変数，文字変数，アドレス変数を指定しており，CHARACTER変数において長さが指定されなかった場合は4文字とみなす．

EVEN

その変数名の先頭番地が偶数番地に割り当てられる．

INITIAL

INITIALは配列あるいは変数に対して初期値を与えるもので，配列に初期値を与える場合には配列要素全部に与えなければならない．初期値は定数を書くのであるがこの定数の属性と変数あるいは配列の属性は一致していなければならない．

INITIALの定数の指定方法は，定数を書くか，数値定数や定数のいずれかであり，後者の場合は定数が数値定数値だけならべて書かれたものとみなす．

＊1. 属性の指定が省略された時にはINTERNAL FIXEDとみなされる．

＊2. 配列名の指定の時にVARYと書くことにより，配列の大きさは一応数値定数で指定され

た通りであるがEXTEND文を使うことにより数値定数単位で配列の大きさを大きくすることが出来る．

＊3　省略形

PROCEDURE → PROC

CHARACTER → CHAR

INITIAL　→ INIT

注　意

PROCEDURE文，DECLARE文は宣言文といい実行文より前になければならない．特にPROCEDURE文はプログラムの先頭でなければならない．

（3）代入文

代入文について述べる前に変数の種類について述べる．

変数 { スカラー変数 { 単純変数 / 配列要素 / 配列

単純変数：その名前を指定することにより一意的にデータが参照できるもの．

配列要素：配列として宣言された場合，個々のデータを参照することは配列の何番目の要素であるかを指定しなければならない．この個々の要素のことを言う．

以上2つをあわせてスカラー変数という．

代入文の一般形

スカラ変数＝スカラ式

配列＝配列；

スカラ式で許されるのはa，b，cが定数，スカラ変数あるいは標準呼び出しとした時に

a＊b＋c

a＊b－c

a／b＋c

a／b－c

a＆b，a｜b，¬a

の以上7種あるいはその一部が省略されたものである．

（4）IF文

IF文はProgrammingにおいて条件分岐をするためにもちいられる．

一般形

IF { 変数 / 定数 / 標準関数呼び出し } [Relational operator { 変数 / 定数 / 標準関数呼び出し }]

THEN GO TO ラベル；

Relational Operatorは次の6種である.

>, >=, =, ¬=, <, <=

Operatorの左右の変数あるいは定数の長さが違う時には短かい方に合せて比較を行なう. その他のIF文使用上の注意は後述する.

(5) GO TO文

GO TO文はProgramのControlを他に移す.

一般形

GO TO ラベル;

(6) 計算GO TO文

単純変数の値によってProgramのControlを移す場所を変えるSWITCHの役割をはたす.

一般形

GO TO 単純変数名, (ラベル1, ラベル2, ····, ラベルn);

単純変数の値が1の時ラベル1, 2の時ラベル2, nの時ラベルnにControl を移す.

(7) DO文

DO文以下このDO文に対する END 文までを変数の値を変えて実行することを指定する.

一般形

$$\text{DO}\quad \text{数値単純変数} = \left\{\begin{matrix}\text{数値定数1}\\ \text{単純数値変数1}\end{matrix}\right\} \text{TO} \left\{\begin{matrix}\text{数値定数2}\\ \text{単純数値変数2}\end{matrix}\right\} \left[\text{BY} \left\{\begin{matrix}\text{数値定数3}\\ \text{単純数値変数3}\end{matrix}\right\}\right];$$

単純数値変数の値を数値定数1 (or 変数1)から, 数値定数3 (or 変数3) くぎりで数値定数2 (変数2)までくり返し実行することを指定する.

＊1. DO文からEND文までをDO Blockと言いこのDO Block内にはDO Blockの外から飛びこんではいけない.

＊2. DO文は入れ子にして指定することが出来るが最大33重までしか指定出来ない.

(8) END文

END文はDO文の終了を示すか, PROCEDURE文の終了を示している. 1つのProgram単位はDO文の数だけそれに対応するENDがあり, PROCEDUREの最後を示す END 文で終っている.

一般形

END;

(9) STOP文

このJOB Stepが完了したことをMonitorに伝える.

一般形

STOP〔数値定数〕；

数値定数はJOB Stepの完了コードを示す．

(10) CALL文

CALL文はもどり番地をSetしてProcedureにControlをわたす．

一般形

CALL Procedure name；

(11) RETURN文

現在実行中のProcedureをCallしたProcedureにControlをわたす．

一般形

RETURN；

(12) ENTER文

以下のプログラム部分がown-codingであることを示す．

一般形

ENTER〔COMMUNICATION MODE〕；

＊1. COMMUNICATION MODEはあってもなくてもよい．

＊2. ENTERにつけられたラベルは無視される．

(13) EXIT文

この文でown-codingのProgramが終了したことを示す．

一般形

EXIT；

＊1. EXITのラベルは無視される．

(14) EXTEND文

EXTEND文はDCL文においてVARYと宣言された変数の大きさ（配列のtotal size）を大きくすることが出来る．変数の大きさは一度EXTENDされるたびにDECLAREされた大きさだけ増加する．

一般形

EXTEND 変数名，変数名，‥‥，変数名；

(15) CANCEL文

CANCEL文はA群 or B群のいずれか一方あるいは両方をProgram上必要がなくなった時にareaをmonitorに返す役割をする．

一般形

CANCEL 群NAME〔，群NAME〕；

群NAMEはAあるいはBである．

(16) ENTRY文

ENTRY文は1つのProcedureに対して複数個の入口を作るためのものである．

一般形

ラベル：ENTRY；

この時のラベルが入口名となる．

(17) 標準関数

17.1 SUBSTR(P1,P2〔,P3〕)

P1：Bit変数

P2：数値定数

P3：数値定数（省略のときは1とみなされる）

P1で指定されたBit変数のP2 BitめからP3 Bitを新しくBit データとして取扱うことを指定する．

例 Aが'10001010110000111111110101100011010'BのBitストリングの時，SUBSTR(A,5,10)は'1010110000'Bという値を持つ．

17.2 ADDR(P1)

P1：変数名

P1の値の入っているAddressを値とするアドレス変数とすることができる．

17.3 INDIRECT(P1)

P1：アドレス変数

この関数の引用によってアドレス変数が持っている番地の値を参照することができる．

17.4 CHARX(P1,P2)

P1：文字変数

P2：数値定数

文字変数P1のP2 Characterめのデータとして扱う．

例 Aの値が'COMPUTER'の時CHARX(A,6)の値は'T'となる．

17.5 CHARY(P1,P2〔,P3〕)

P1：文字変数

P2：数値定数

P3：数値定数（省略のときは4とみなされる）

文字変数P1のP2 CharacterめからP3 Characterをデータとして扱うことを指定する．ただしP2＝4n+1，P3＝4n（nは自然数）の値を持たなければならない．

17.6 SHIFTR(P1,P2)

P1：スカラ変数でかつ1word分しか指定できない．

P2：数値定数

スカラ変数P1の値をlogicalにP2 Bit右シフトをし，左側に0をつめる．

17.7 SHIFTL(P1, P2)

P1 ： スカラ変数でかつ1word分しか指定できない．

P2 ： 数値定数

スカラ変数P1の値をlogicalにP2 Bit左シフトし右側に0をつめる．

注 意

標準関数SUBSTRはその値として第1 Bitめにつめた値をとっているため，大小比較は出来ない．その必要のある時はSHIFTRを使う．

2．PLDの使用経験

実際に当センターでPLDを使ったシステムはCPLとFACOM 230−60 FASP のオートフローがある．CPLに関していえばCPLはコンパイラフェーズとランタイムフェーズにわかれているがコンパイラフェーズの90%以上（カード枚数で）がPLDでコーディングしてある．PLDではコーディングがしにくくアセンブラにたよったところは次のような部分である．

① MONITOR−Vとのリンク

② 10進演算，浮動小数点演算

③ 入出力機能

④ 実行アドレスの参照が必要な場合

ランタイムフェーズに関してはほとんど PLD は使用していない．その一番大きな理由は，レジスタの扱いが現在の PLD ではやや不足しているせいである．CPL がランタイムのために作り出す情報は機械語の部分とインタープリターへのパラメータの部分に分かれる．インタプリタ部分は PLD で書くことは原則的には可能であるが，機械語で実行した結果がレジスタにはいっている場合そのレジスタをそのまま利用しているために PLD とのつなぎができなくなっている．

オートフローの場合には，PLD の使用を強制しなかったこと，作りあげるメンバー4人はすべてアセンブラをよく知っていたこともあって，2人がそれぞれ50%位を PLD でプログラムしたにとどまった．その理由としては

① アセンブラを知っているのに新しく他の言語を覚える必要はない．

② アセンブラにくらべるとオブジェクトの効率が悪い．

③ リエントラントにする必要がないので PLD の大きな特長の1つが生かせない（PLDはオブジェクトをリエントラントにもノンリエントラントにもできるが）ということである．

このように，PLD の存在が必ずしも全面的に受け入れられたわけではないが，ともかくもこの1年半の経験から使用者の評価をまとめてみた．ただ，使用経験者のすべてはアセンブラ言語によるコーディングの熟達者であり，たえずアセンブラ言語との比較において評価がなされたのは止むを得ぬことである．

PLDの長所

① コーディング時間が短縮できる．

② プログラムが読みやすい．

③ 不注意なミスが少くなる．

④ アセンブラがPLDの中で自由に使える．

PLDの短所あるいは要望

① 一般的に言ってオブジェクトプログラムステップ数が多くなる．特に文字処理においていちじるしい．

② レジスタ等の管理をユーザー側ででも出来るようにして欲しい．

③ ディバッグ機能が少ない．

④ 入出力機能が不足．

⑤ データにDECIMAL，FLOAT 属性およびその間の相互変換の機能をつけ加える．

⑥ 同じエリアを異った属性で使いたい．

⑦ ONコンディションの追加．

PLDは短期間（言語仕様の検討から始めて4ケ月間，工数は約12人月）で作ったいわば即席言語であって色々な欠点を持っている．特に使用者はすべてアセンブラ言語を知っている人間であるという前提にたっていたのでなによりもオブジェクトの効率に留意した．また実際にPLDの作り出すオブジェクトプログラムのパターンも明示した．そして言語仕様の中にアセンブラ言語を書くことを許すことにより，PLDの持っていない機能，あるいは効率がよくないところはアセンブラでプログラムを書く余地を残した．これがいいとか悪いとかは別としてアセンブラ言語をよく知っているものには歓迎された機能である．将来PLDの経験から新しい言語を作る場合にも残しておきたい機能だと考えている．

今回のPLDの使用経験から，ビットハントリングおよびキャラクタハンドリングの便利な機能を持ったコンパイラ言語ならば，基本的にはコンパイラ作りに向いていると考えてもよいのではないかと感じた．しかしながら，実際にシステムプログラムを作成する者はコンパイラが使用するメモリサイズやテーブル類の大きさ等をなるべく少くするよう考慮しながら作るのでオブジェクト効率がかなりよいものであることを証明しない限り使う気にならないこと，そしてシステム設計あるいはプログラム設計時においてどのような使い方をすれば効率がよいかという言語の特長を生かすと生かさないのとではかなり製品の質が変ってくる．したがってそういった細かい点も心得た上でシステム記述用言語を使用する必要があろう．

3．オブジェクト効率とディバッグ機能

以上のような使用経験から記述言語の性質として特に重要だと感じたのは，やはりオブジェクトの効率をできるだけあげること，そしてソース言語のレベルでのデバッグをやり易くするという2点であった．

そこでその2点について少し考えてみよう．

一般にコンパイラレベルの言語を使用してシステムプログラムを作る場合に主たる目的となるのは

① 作業効率をあげる．

② マシン・インディペンデントにする．

のいずれか一方を満たしていることが必要となる．

いずれにせよ，現時点においてコンパイラレベルの言語で作り上げたシステムは，アセンブラレベルの言語で作ったそれよりもオブジェクトの実行の効率がおちている．この効率をどの程度までよくすればよいのかはそのシステムの用途とハードウエアの環境によるが，いずれにせよ実行効率をよくすればする程使われ易いと考えてよい．しかしながら通常のコンパイラ言語の仕様ではオプティマイズするのは非常に大変なことである．しかしプログラマーがアセンブラの知識を持っていればコンパイラーがオプティマイズしたプログラムよりもさらによいプログラムを比較的簡単に作ることができる．この点からシステム記述用言語は本来のプログラムロジックを記述する部分と記述言語のコンパイラがオブジェクトプログラムを作りだすための補助情報をプログラマ側からコンパイラに知らせるオプションの2つをもうけておけば共通言語としてのシステム記述言語と計算機のハードウエアにディペンドすることによって効率をよくする部分との両立が可能になる．

また初心者はレジスタ類の有効な使用法に対する知識がうすいから，プログラムロジックだけを記述するだろうし，熟練者はオプティマイズに気をくばりたければオプションを使うであろう．そうすることによりよりよいオブジェクトを作りだすことが出来る．

ディバッグ機能については，各モジュールの段階のディバッグと各モジュールをつなぎあわせた段階におけるディバッグの2つの段階にわけることができる．

モジュール段階のディバッグにおいてはそのモジュール内をどのようにコントロールが移っていったかという情報の適当な時点でのメモリダンプがあればよい．インテグレーションの段階においては各モジュール内をどのようにコントロールが移ったかという情報と，パラメータとして，どのようなものがわたされたかを知る必要がある．まとめてみると次の様な機能を持っていればよいことになる．

① モジュール内のコントロールがどのように移っていったかを印刷する．

② モジュール間のコントロールの状態．

③ SNAP SHOT

④ 受け渡す（あるいは受け渡たされた）パラメータのダンプ．

⑤ ランタイムにエラーが起った場所を示すこと．

以上の機能のうち，SNAP SHOTだけは前もってユーザーがプログラム中で指定する．これらのディバッグ機能はコンパイルタイムにディバッグモードを指定し，さらに，ランタイムにプリントアウト指定があった時のみに働けばよいことにすればよい．

4. PLDとアセンブラ言語の教育実験

ハイレベルのシステム記述用言語を必要とする目的の1つは，比較的初心者が容易に使いこなせるようにということである．

PLDがそういう目的をうまく果し得たかどうかという事は当センターの経験では残念ながら確信を得ることが出来なかった．ということはこのPLDを利用したCPLやオートフローの作成者は先程も述べたようにすでにアセンブラ言語の熟練者であり，PLDを憶えること，あるいは使いこなすことは，何等問題はなかった．しかし逆にPLDが，初心者向きであったかどうかという実験の対象とはなり得なかったこととなる．

そこで同じレベルの初心者（FORTRAN位は知っているが，実務経験はほとんどない）を10名選び2つのグループに分け1つのグループにはPLDを，他のグループにはアセンブラ言語をそれぞれ3日間づつ教えた．そして3日目の午後適当なプログラムをそれぞれ作らせて見たが，PLDグループはコンパイラのシラブルリードの1部を作るまでになったが，アセンブラ組は積和のプログラムが出来る程度にしかならなかった．

アセンブラ言語が初心者にむずかしく感じられるのは次の様な点である．

① 変数名とアドレスの対応をつけること．

② 命令が基本的すぎてどのように組合わせてプログラムを作るかがわからない．

③ レジスタの有効利用

きわめて小規模な実験ではあったが，PLD程度の簡単な文法をもったコンパイラ言語なら，新人であっても充分システムプログラムの作成が出来るということがわかった．

おわりに

システム記述言語は，その使用目的から考えて，計算機が出来ると同時期に出来ている必要があり，この点については，ベイシックアセンブラと同じような役割を持っていると考えてよい．したがって，始めから大規模なシステムにすることは事実上無理である．むしろシステム記述用という単一目的をもったなるべく簡単な仕様にすべきであるかも知れない．

しかしこれはなかなかむずかしい問題である．言語仕様が簡単でしかもオブジェクト効率が充分吟味され，その上初心者でもうまく使いこなせるというこの矛盾した要求をどういう形で解決するかなお検討を要する問題である．