解　説

# システム記述用言語*

筑　後　道　夫*

## 1. まえがき

情報処理の普及につれて，大規模な，高度な，または多様な計算機システムが必要となり，高性能のハードウェアと共にシステムを構成するソフトウェアの開発を能率化することが益々必要になっている．

これらのソフトウェアは大別すると，①システムの中核となるオペレーティングシステム（主として制御プログラム，OS と略記する），②ソフトウェアの作成手段となるコンパイラやユーティリテイプログラム，および，③実際にシステムの業務を行うプログラムに分けられる．これらのうち①と②はシステムの運転やプログラミングのために共通に使用されるもので一般にシステムプログラムと言われている．専用のリアルタイムシステムにおいては③のプログラムの一部は OS と密接な関係をもっているのでシステムプログラムと言うことができる．

これら各プログラムのうち③の大部分は，COBOL，FORTRAN などの高水準プログラム言語を用いてプログラミングされているが，システムプログラムは，システムの性能を左右するということで，特に商用システムにおいては，アセンブラ語でプログラミングされている場合が多い．しかし，各種計算機システムの開発に必要とされる大量のシステムプログラムを，短期間に，容易に，しかも経済的に製造することが益々必要とされ，このため各種のシステム記述用言語とそのコンパイラが開発され，普及され始めている．

ここで記述という意味は，対象とするプログラムの論理を特定のプログラム言語で表現すれば，その言語のコンパイラが直接機械語のプログラムを生成するという意味である．一方，システムプログラムの設計を支援するという立場から，OS の設計用言語[1),2)]やコンパイラの自動作成手法[3)]が研究されており，これらの技術もシステムプログラム開発の効率化をもたらすわけであるが，本稿では，アセンブラ語に代って，プログラムの生産性，保守性を向上させるため高水準のプログラム言語を導入して，直接プログラムを記述・製造している代表例と共通的技術について解説を行う．

## 2. システム記述言語の必要条件

システム記述言語の目的は，アセンブラ語の長所をもち，欠点を克服してシステムプログラムの生産性を向上させることであるが，高水準言語をシステム記述用に使用するときの長所と短所をのべ，その必要条件を考える．

長所としては

（1） プログラム製造の所要工数と期間の削減

プログラムの製造工程は，設計，コーディング・デバッグ，検査に大別され，後の2工程で全体の 60% 前後の工数を要し，また，プログラムの年間生産高は，使用する言語の種類によらず一定であるともいわれている[4)]．また，システムプログラムをアセンブラ語と高水準言語を用いて記述した比較実験では，アセンブラ語を使用した場合に比して，ステートメント数は 2/3 に，工数は 1/2 に減少している[5)]．これらの例から，高水準言語を用いることにより，見かけ上のプログラムの規模が縮少し，したがって所要工数や期間が少なくてすむ．

（2） プログラム全体の理解が容易になり，改良や保守が容易になる．

高水準言語で記述するとプログラム全体の構造がつかみやすく，各ステートメントの意味も理解しやすい．一方システムプログラムは，一度完成した後の改良や，運用時の欠点の修正がどうしても必要であり，またこれらの作業は最初の製造にタッチしたプログラマが，全て残っていて担当するとはかぎらない．したがって，プログラム全体の理解が容易であるというこ

* System Description Language, by Michio CHIKUGO (N. T. T., Yokosuka Electrical Communication Laboratory)
** 日本電信電話公社横須賀電気通信研究所データ処理研究部

とは非常に重要である．Multics のOS の開発において，PL/I を記述言語として用いる事により，多くの担当者が変りながら，OS の改良を重ねることが出来たのも，PL/I プログラムの理解の容易さが大きく貢献しているといわれている[6]．

（3） 異なった計算機システムへのプログラムの移行が期待出来る．

OS などは，ハードウェアのアーキテクチャに依存する部分が多く，直接的な移行にはエミレーション等他の技術が必要であるが，コンパイラ等は，ハードウェアや OS に依存しない部分が大きいし，またそのように意識して作れば，他のシステムでも使用出来る．即ち，高水準言語で記述されたプログラムを移行システム上で動作させるためのコンパイラを準備するだけでよい．

一方高水準言語をシステム記述言語とする場合の欠点としては

（4） システムに必要な性能が得にくい．

一般に FORTRAN や PL/I などの高水準言語は，同じプログラムをアセンブラ語で記述する場合にくらべ，オブジェクトプログラムの実行時間や，メモリスペース所要量が大きくなる．これら性能低下の割合は，言語やコンパイラの作り方にも依存するが，その程度は，リアルタイムシステムのプログラムモジュールの記述実験によれば，FORTRAN と PL/I を用いた場合，アセンブラ語記述に比して2～7倍の実行時間を要している[7]．

（5） ハードウェアの機能に十分アクセスできない．

ハードウェアの特権命令や，レジスタなどハードウェアを直接アクセスする機能が不足している．

上記の長所を保持し，欠点を補なう機能がシステム記述言語の必要条件であり，またこれら条件の具備状態がシステム記述言語の評価尺度ともなる．

## 3. システム記述の適用例

1960 年代の後半から 70 年代の前半にかけて，各種システム記述用言語の開発や適用が行われて来た．即ち，FORTRAN や ALGOL タイプの言語，および，PL/I のフルセットまたはサブセットが適用された．対象となるシステムプログラムとしては，ハードウェアと独立なコンパイラの記述が多く試みられ，一部は商用機にも適用されている．そのほか OS を対象に，アセンブラ語との共用もしくは機械語機能を付加した言語が試みられた．以下，米国の例[8]，我が国の例[1]で，内容が比較的よく報告されており，かつ各言語のタイプのうち代表的なものを表-1 に示し，その概要をのべる．

表-1 システム記述言語の適用例

| 言語名 | 適用状況 コンパイラ | 適用状況 OS | ベース言語 | 適用システム | 実施機関 |
|---|---|---|---|---|---|
| Extended ALGOL, DCALGOL, ESPOL | ○ | ○ | ALGOL 60 | B6500/6700 MCP | Burroughs |
| FORTRAN | ○ | ○ | —— | Honeywell 516, 832 | NASA |
| LRLTRAN | ○ | ○ | FORTRAN | CDC 6600/ 7600 TSS | Lawrence Radiation Lab. |
| PL/I | ○ | ○ | —— | Multics | MIT |
| SABRE PL/I | | ○ | PL/I | SABRE 360 | —— |
| PL/S | ○ | ○ | 〃 | OS/360 | IBM |
| BPL | ○ | | 〃 | NEAC | 日電 |
| PL/I＊ | ○ | | 〃 | 大形プロジェクト計算機 | 大プログループ |
| PL/IW | ○ | ○ | 〃 | HITAC 5020 TSS | 日立 |
| SYSL | ○ | ○ | 〃 | DIPS-1OS | 電電公社 |
| PL 360 | ○ | | —— | IBM 360/370 | —— |
| SL 45 | ○ | ○ | —— | FACOM 230/45 | 富士通 |

(注) ○印は大部分もしくは，いくつかのモジュールが記述されたことが報告されている．

（1） ALGOL

B 6500/6700 の MCP (Master Control Program) の開発に，ALGOL を拡張した Extended ALGOL が使用された．この言語は，ALGOL 60 にビット処理，ストリング処理，リスト処理，イベント処理，非同期処理等の機能を追加したものである．MCP の機械依存部は，機械へのアクセス機能をもつ同族の言語，ESPOL で記述された．

（2） FORTRAN

NASA で，Honeywell 516, 832 の OS と支援システムの開発に FORTRAN を使用した．この場合は FORTRAN を変更せず，I/O とのインターフェイス部や効率を重視するルーチンのみはアセンブラ語を用い，文字処理は，1文字を1語とし integer および integer array としてあつかった．

その他 FORTRAN をベースに必要な機能追加を行った言語に LRLTRAN があり，これにより CDC 6600/7600 による TSS システムを開発した．追加し

た機能は，BYTE, STRUCTURE, ビットやシフト機能，アドレス操作などである．

（3） PL/I[6),9)]

GE 645 を用いた TSS システム Multics の記述に Multics PL/I を用いた．本コンパイラの言語仕様は，1968 年の IBM 仕様と大体同じである．PL/I を選んだ理由は，モジュラリティ，豊富な機能，機械独立性の利点があるからである．データ・ベースや特権命令のサブルーチンは性能を重視してアセンブラ語で記述した．全体の規模は 1500 モジュールで内アセンブラ語のモジュールは 250 である（1 モジュールは平均して 200 の PL/I 文）．またトラヒックコントロール，割込処理，ページフォルト処理の一部は性能改善のため後にアセンブラ語でリコーディングした．保守性を良くするため逆に PL/I でリコーディングしたものもある．PL/I コンパイラのオブジェクトプログラムは，アセンブラ語記述に比して 2 倍くらい効率が劣るが，これはコンパイラの最適化処理が十分でないことが大きな原因であるとしている．

（4） PL/I のサブセット

PL/I はシステムプログラムを記述するのに適した豊富な機能を有するが，フルセットの言語仕様ではコンパイラの開発に多大の工数を必要とすること，およびオブジェクトの効率が悪くなることの理由で，システムプログラム記述に必要な最小限の機能のみを選んだサブセットや，さらにその上にいくつかの機能の追加や変更を行った言語が米国と我が国で数多く開発されている．

米国における例としては，IBM のユーザの開発したものとして，コンパイラ記述用に使用された XPL, American Airline の座席予約システムのモジュールの記述に使用された SABRE PL/I[10)]，IBM 社内のシステムプログラム用に開発された PL/S[15)] などがある．PL/S は，アセンブラ語を PL/S プログラム中にインラインに書けること，構造体のマッピングに自由度をもたせていることなど後述する（5）項中間水準言語の性格をもっている．そして OS/360 のいくつかのモジュールの記述やコンパイラの記述に用いられている．

我が国では，日電のコンパイラ記述用の BPL[11)]，電総研の大型プロジェクトでコンパイラ記述用の PL/I*，日立の HITAC 5020 TSS の記述に用いられた PL/IW[12)]，通研の DIPS-1 OS の記述に用いられている SYSL[5)] およびその他の社にも同様の言語が開発されている．

これら PL/I のサブセットやそれに一部機能の変更を行った言語は，オブジェクトプログラムがアセンブラ語記述に近い性能が得られており，実用システムにも適用されている．内容については 4 章で詳細にのべる．

（5） 中間水準言語

ハードウェア機能への直接アクセスとオブジェクトプログラムの高能率性を保ち，加えて ALGOL など高水準言語のスタイルをもって書き易く読み易いという目的で開発された言語に PL 360 がある．即ちアセンブラ語と高水準言語の中間的な水準の言語である．

PL 360 は IBM 360 上で ALGOL-W コンパイラの記述に適用された．言語の主要な特徴は，Procedure, If then else, for ループ，array など ALGOL 的文の他に，レジスタや機械命令が直接に書け，オブジェクトコードとステートメントとの対応も明確になっている．

この流れをくむ言語は，我が国でもいくつか試みられている．SL 45 は，PL/I のスタイルを導入したもので，FACOM 230/45 上で，PL/I コンパイラや通信制御プログラムの記述に適用され，ソースプログラム文数もアセンブラ語記述に比して半分近く少なくなり工数削減に役立った[13)]．

## 4. PL/I サブセットの言語仕様とコンパイラ

### 4.1 共通的言語仕様

システムプログラムを記述するに必要な機能としては，ビット列や文字列処理，リスト処理，表操作，記憶域割付などが特徴的であり，この他 OS 記述に対してはレジスタや機械命令を使用出来るための機能が必要である．3 章にのべた PL/I サブセットの言語に共通的な言語仕様を**表-2**（次頁参照）に示す．

これらの言語仕様を用いて，コンパイラや関連するユーティリティプログラムはほとんど記述出来る．ただし，特有の入出力操作やランタイムルーチンなどで OS の特殊機能に依存するものはアセンブラ語で記述し，システム記述言語で書かれたプログラムからコールする．

OS を記述の対象とする場合は，ハードウェアとのインターフェイスや入出力動作などの記述が必要である．このためシステム記述言語に次の様な機能をもたせると便利になる．

表-2 PL/I サブセットの共通的言語仕様

| 言語仕様の分類 | | 言語仕様の内容 |
|---|---|---|
| 大分類 | 小分類 | |
| データの型 | 算術データ | BINARY FIXED, 10 進整定数 |
| | 列データ | CHAR, BIT, 各々の定数 |
| | 制御データ | LABEL, ラベル定数, POINTER, AREA |
| データ集合 | | 配列（次元1），構造体，構造体の配列 |
| 記憶域クラス | | STATIC, AUTOMATIC, BASED |
| その他のデータ属性 | | INITIAL, ALIGNED, UNALIGNED, CELL, DEFINED |
| 演算子と式 | 演算子 | 算術，ビット，比較，ポインタ修飾 |
| | 式 | データの型の混合を禁止 |
| 組込関数と擬変数 | 列処理 | SUBSTR, UNSPEC（両者は擬変数にもなる） |
| | リスト処理 | ADDR, NULL |
| | 型変換 | BIT, CHAR, BINARY |
| 文 | プログラム構造文 | PROCEDURE, END, DO, ENTRY |
| | プログラム制御文 | GOTO, IF, CALL, RETURN, STOP, 空文 |
| | 記憶域割付け文 | ALLOLATE, FREE |
| | 入出力文 | OPEN, CLOSE, READ, WRITE |
| | 割込み，デバッグ文 | ON, PUT DATA |
| | 定義文 | DECLEAR |

（1） アセンブラ語がインラインに任意に書ける機能[14),15)]

これによりアセンブラ語記述プログラムとのパラメータの受渡し規則などを考慮してコールするよりコーティングが容易になるし性能もよくなる．例えば

```
DCL WORK CHAR (7);
      ⋮
CODE;
   & SVC 15, 15 (,0)
   & MVI WORK, 48;
      ⋮
ENDCODE;
```

CODE ブロックが1つのステートメントの様にそう入される．

（2） レジスタや特殊な作業域の管理をプログラマに指定させる機能[14)]

OS の SVC ルーチンなどでは，その PROCEDURE のプロローグ，エピローグ処理のレジスタの退避・回復や作業域を PSECT にとるか CSECT にとるかなどの指定をプログラマが指定出来ることが必要となる．例えば，

```
A: PROC OPTIONS (BR(I-J)
     GR(K-L) FREENV)
```

と指定した時，BR レジスタの I から J，GR レジスタの K から L まではコンパイラが退避・回復処理を省略して他の目的に使用してもよいことを FREENV オプションで指定する．プログラマがこれらのレジスタを A: PROC で使用しないならば退避・回復処理のロスがはぶけるし，使用する場合は，そのレジスタだけを退避・回復すればよいので効率のよいオブジェクトプログラムが得られる．

### 4.2 コンパイラの最適化項目

コンパイラが効率的なオブジェクトコードを生成することの出来るケース（最適化項目）を判断して，コード生成を行う事は，FORTRAN, PL/I などの高水準言語でも行うのが普通であるが，システム記述言語の場合には，この最適化をきめ細かに行う必要がある．これら最適化項目の主なものについてのべる．

（1） 共通式の演算を重複して行わない．

$$A=B+C\times D$$
$$H=I-C\times D$$

というコーディングのとき，2番目の式の C×D の演算を行うコードの代りに，一番目の式の C×D の結果をそのまま用いるコードを生成する．

（2） DO ループ内の制御変数を添字として持つ変数の番地計算

配列要素 A (I+定数) が制御変数 I を変化させてくり返し参照される場合，配列要素の番地計算をインデックスレジスタの加算だけで対処するものである．

（3） 使用頻度の高い変数に対してベースレジスタあるいは汎用レジスタを固定して割当てる[5)]．

プログラム中で何度も出てくる算術データやポインタ変数および，DO ループの制御変数を汎用レジスタやベースレジスタに固定して割当てておけば，変数の参照時にレジスタのロードやストアが不要になる．この場合コンパイラは，静的な使用頻度を見て判断するわけであるが，さらに，言語仕様に REGISTER 属性を導入して[14),15)]，プログラマが動的な使用頻度も考慮して，変数をレジスタに固定するように指定させる場合もある．

（4） IF 文での不要な分岐命令コードを生成しないようにする．

（5） 高速の機械命令コードを生成するようにする．

## 5. システム記述言語適用上の問題

### 5.1 評　価

2章でのべたシステム記述言語の条件に照らして，

各実施例から記述性，性能，生産性について評価してみる．

（1） 記 述 性

記述出来るということは，コンパイラにより生成されたオブジェクトプログラムの性能やプログラムの生産性も妥当な範囲に入るという意味で評価する必要がある．

FORTRAN や ALGOL タイプの言語ではシステム記述に必要な言語仕様を追加しているが，PL/I タイプの言語ではこれらの追加機能はおおむねもっている．PL/I タイプの言語の場合，コンパイラの 90% (SYSL)，OS については，80% (Multics)，60% (PL/IW)[16] という事が報告されており，他の部分はアセンブラ語で記述したルーチンのコールやインライン展開で対処出来る．

その他，OS 記述上特徴的な点は，1つのデータを異なる属性でアクセスする必要が出てくる事である．

このような例は，ある変数をビットパターンとして見たり，アドレスとして見たりする．このとき CELL 属性で2つの属性を与えたり[16),17)]，

```
DCL 1 SFIELD CELL,
      2 A PTR,
      2 B BIT (32);
```

または，UNSPEC 擬変数を使用したりする．

```
DCL C PTR;
DCL D BIT (32);
  UNSPEC (C)=D;
```

（2） 性 能

アセンブラ語記述に比して，実行時間やメモリスペースがどのくらいかという比較値で表わされる．そして，それは言語というよりコンパイラの性能にほとんど依存する．

コンパイラのモジュールでの比較実験で実行時間比 1.11，オブジェクト・サイズの比 1.17 (SYSL)，OS の記述では実行時間比約 2 (Multics)，オブジェクトのサイズ比 1.5 (PL/IW) と報告されている．Multics PL/I の経験では，サブセットにしてもっとコンパイラを良くしたいう意見であり，また PL/IW では研究所での実験ということで工数を節約した．したがって PL/I のサブセットの言語でコンパイラの評価・改良を行えばさらに性能は向上すると言える．

（3） 生 産 性

アセンブラ語記述の場合と直接比較した値では2倍生産性が上がるという報告がある．その他の使用経験からもコーディング・デバッグが容易で，ドキュメンテーションが非常によくなりかつ大きなシステムプログラムを少ないプログラマで製造・改良・保守が出来るので，ソフトウェア技術の向上やプログラマの地位の向上にも貢献すると思われる．

### 5.2 効率的コーディング技法

プログラムはコーディングの仕方により，見やすさ，誤りにくさ，ひいては生産性に大きな影響を及ぼし，ストラクチャード・プログラミングが注目されているが，さらに，高水準言語を使用する場合はオブジェクトプログラムの効率や記述性にも大きな影響があることがしばしば経験されている．コンパイラは 4.2 でのべた如く，種々の最適化処理を行ってはいるが，それには限度がある．即ち，コンパイラが複雑になり，翻訳時間を大きくしたり，コンパイラの製作や正確さの保証を困難にしたりする．そこで効率のよいオブジェクトプログラムを得るコーディング技法が必要になる．

Multics における PL/I 使用の経験によれば，システム全体の見通しが良くなり，したがってシステムが出来上がってからネックとなっている点を見出し，Global な観点からの性能改良が容易であったといわれている．またシステムを構成するモジュールやルーチンが特に高頻度で実行される場合，その性能がシステム全体の性能に影響する．

したがって，コーディング時に効率的な技法を心掛けると共に，一度完成したシステムの動的な評価分析を行い重点的な部分のコーディングの改良が行われる．これらコーディング技法は，コンパイラの最適化処理のやり方にも関係するので一般的にいう事は困難である．これに関して IBM 社の PL/I コンパイラを中心とした技法例が紹介されているが[18]，次に筆者等の経験から得られたいくつかの例をあげる．

（1） PROCEDURE のプロローグ，エピローグ処理を少なくするようコーディングする．

（i） プログラムを多くの手続きに分割せず出来るだけ大きくまとめる．

（ii） 頻繁に使用される共通ルーチンは，実行効率を重視するときは，各々使用する手続の内部に内部手続として入れる．

（iii） もし，複数の共通ルーチンがあるときは，これらをまとめて1つの外部手続きにすればスペース効率は良いが実行効率は悪いので，1つず

の独立した外部手続きとした方が良い.

(2) 頻繁に使用される変数がレジスタに割付けられやすいようにコーディングする. これはコンパイラが4.2のような最適化処理を行っている場合に有効となる. 例えば, 複数の EXTERNAL データやパラメータをまとめて1つの構造体として宣言する.

```
DCL (A,B,C) BIN EXT;
```

の代りに

```
DCL 1 EXTDATA EXT,
    2 (A,B,C) BIN;
```

とすると EXTDATA にベースレジスタが優先的に割付けられ, EXTDATA.A でAが参照される.

(3) 多分岐の場合にはラベル配列を使用する.

```
IF A='00'B
   THEN GOTO L(1)
    ELSE IF A='01'B
         THEN GOTO L(2)
          ELSE .....
```

というようにビットAの値により飛び先が変るとき

```
DCL L(4) LABEL;
GOTO L(BINARY (A, 15)+1);
```

とコーディングする方がコーディングも簡単であるしオブジェクトコードもよくなる.

(4) ALIGNED と UNALIGNED 属性の使いわけ.

配列や構造体に指定された場合, ALIGNED のときは, データが語境界で区切られるのでメモリスペース効率は悪いが, オブジェクトコードは少なくなり実行効率は良い. UNALGINED のときはメモリスペースはパックされて使用効率はよいが, 実行効率は ALIGNED より悪くなる.

したがって, 実行効率とスペース効率のどちらを重視するかを判断して選択する.

(5) 高水準言語の有する機能でコーディングの発想をする.

アセンブラ語記述を前提とした設計は, しばしば高水準言語での記述を非能率にすることがある. プログラムの設計時に言語の特徴を生かすよう心掛け, アセンブラ語でなければ記述出来ない部分を少なくする. また, アセンブラ語による特殊処理をマクロ化して共用をはかるようにすると, 生産性も向上するし, ドキュメンテーションも良くなる.

## 6. むすび

システムプログラムも, 高水準言語で記述出来るという種々の試みが積み重ねられ, またコンパイラの効率も良くなって来たので, 記述出来るかどうかという段階を経て, いかに記述するとよいかという事が問題にされる段階に来ていると言えよう. 言語仕様としては, PL/I から必要な機能を選択したサブセットで対処出来ることが多くの経験から明らかになっているが, 実用上からさらにいくつかの機能もとり入れられている. 一方, 商用システムにおいても, ハードウェア技術の進歩を反映して, アセンブラ語記述との比較による性能を問題にするより, ソフトウェアの生産性を向上させて, よりよいシステムを作る技術に重点が移っていると思われる. また研究としては, システムプログラムの設計段階まで記述水準を上げた言語がいろいろ提案されて来ており, 文字通りシステム記述言語になることを期待したい.

## 参考文献


1) 竹下亨: 日本におけるプログラム言語, bit, Vol. 6, No. 9, p. 925 (1974).
2) 林達也: システム設計言語 DEAPLAN について, 情報処理, Vol. 14, No. 9, p. 652(1973).
3) 井上謙蔵: コンパイラ自動作成の諸方法, 第12回プログラミングシンポジウム報告集, プログラムシンポジウム委員会, 情報処理学会, p. 31 (1971).
4) クラーク・ワイスマン: 最近のソフトウェア技術の展望と将来, 情報処理, Vol. 14, No. 10, p. 742 (1973).
5) 寺島信義: DIPS-1 における高能率システム製造用言語の実用化, 情報処理, Vol. 16, No. 6, p. 499 (1975).
6) F. J. Carbató: PL/I as a tool for system programing, Datamation, p. 73, May 1969.
7) R. E. Merwin: Programing language efficiency inrealtime software system, Proc. SJCC p. 155 (1972).
8) J. E. Sammet; Brief survey of languages used in system implementation, ACM SIGPLAN, Vol. 9, No. 9 (1971).
9) F. J. Carbató, et. al: Multics-The first five years, Proc. AFIPS, p. 571 (1972).
10) M. Hopkins; SABRE PL/I, Datamation, p. 35, Dec, 1968.
11) 小久保靖世, 他: コンパイラ記述用言語 BPL, 情報処理, Vol. 11, No. 6, p. 342 (1970).
12) 中田育男: HITAC 5020 TSS の記述言語としての PL/I サブセット, 昭43年連合大会, 2529 (1968).
13) 後藤真美, 他: システム製造用言語 SL 45, 11回情処大会, 132 (1970).

14) 寺島信義, 他: システム製造用言語SYSL-2の設計, 情報処理, Vol. 16, No. 8, p. 692～697 (1975).
15) G. Wiederhold, et. al; Inferred syntax and semantecs of PL/S, ACM SIGPLAN, Vol. 9, No. 9 (1971).
16) 浜田穂積, 他: PL/IW によるシステム開発, 第 11 回プログラミングシンポジウム報告書, プログラムシンポジウム委員会, 情処学会 (1970).
17) 永瀬淳夫: システム製造用言語の適用上の問題, 昭 50 年電子大会, 1310 (1975).
18) 竹下亨: PL/I-複合プログラミング言語, p. 317. 日本経営出版, (1970).

(昭和50年6月2日受付)