資　料

DIPS-1 における高能率システム製造用言語の実用化*

寺　島　信　義**

Abstract

SYSL-System Description Language, which is a higher level language like FORTRAN, COBOL and so on, has been developed as a system implementation language for use by DIPS-1 operating system, a large scale OS for time-sharing services, banking control services and so on.

Run-time efficiency and space efficiency of SYSL object codes are 1.11 times and 1.17 times, respectively, of those obtained from assembly language.

SYSL has been used to write about 90 per cent of language processors and other processing programs of DIPS-1 OS.

SYSL has reduced the huge amount of time required for coding, debugging, maintaining and documenting.

1. まえがき

DIPS-1[1)2)] (Dendenkosha Information Processing System. Model 1) の高能率システム製造用言語（以後 SYSL と称す）は，DIPS-1 のオペレーティング・システム (OS) を記述・製造するために開発された高水準言語である．DIPS-1 は，電電公社におけるデータ通信事業の一環として開発された超大型電子計算機である．これらのサービスを提供するための OS は，超大型なものとなる．このような超大型 OS の製造の工期短縮および製造，保守，訓練の一元化などをねらう一つの手段として，高水準言語で，システム開発を行なうことを試みた．この場合問題となるのは，目的プログラムの実行時間および記憶域の低能率性である．そこで，われわれは，目的プログラムの高能率化および SYSL の早期開発を目的として，まず適用領域を限定し，FORTRAN, COBOL, PL/I, アセンブラなどの言語処理プログラム，リンケージ・エディタ，デバグ・ルーチン，浮動小数点演算を伴なわない処理プログラムなどを主たる対象として，PL/I[3)] からサブセッティングを行ない，性能向上に必要な機能追加を行なった仕様を制定し，種々の最適化技法をとり入れ，実用化を行なった．

本論文では，SYSL の言語構成，最適化技法および評価について述べる．

2. 言語構成

SYSL は，基本的には，システム・プログラム記述に必要な機能を備えた PL/I のサブセットである．したがって SYSL の構文則，意味則は，PL/I と同じである．サブセッティングに当り，コンパイラおよび得られる目的プログラムの性能に悪影響を及ぼす仕様を，記述能力を損なわない範囲で制約してある．また目的プログラムの性能向上に寄与する属性を追加してある．つぎに，SYSL の特徴的な仕様概要について述べる．特に記憶域属性および ON 文に関する制約事項については，処理方式の観点より明確にする．

2.1 データの種類と構造

SYSL のデータには，BINALY FIXED, BIT, CHARACTER, POINTER, OFFSET, AREA および LABEL データがある．このほかに入出力文のファイル属性として FILE, SEQUENTIAL, etc. がある．

データ構造には，スカラー，配列および構造体がある．ここにスカラーとは単一のデータ項目，配列は homogeneous なスカラーからなる 1 次元の集合体，構造体はスカラー，配列あるいは構造体からなるデー

* DIPS-1 High-Efficient System Description Language by Nobuyoshi TERASHIMA (Processing Programs Section, Yokosuka Electrical Communication Laboratory, N. T. T.).

** 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

Table 1 SYSL language specifications summary

1	Language Character Set	49 character set	
2	Data Elements	Scalar items Arrays. . . . . 1 dimension Structures. . . . . up to 8 levels	
3	Data Attributes	Arithmetic data	BIN FIXED data
		String data	CHAR, BIT data
		Locator data	POINTER, OFFSET data
		Label data	LABEL data
		Area data	AREA data
4	Statements Classification	Assignment statement	
		Control statements	GO TO, IF, CALL, RETURN, STOP statements
		Data declaration statement	DCL statement
		Error control and debugging statement	ON statement
		Input/output statements	OPEN, CLOSE, READ, REWRITE, WRITE, LOCATE statements
		Program organization statements	PROC, END, DO, ENTRY statements
		Storage allocation statements	ALLOCATE, FREE statements
		Null statement	

タの集合体である．これらの構成を **Table 1** に示す．

2.2 プログラム構造

（1） 文

文は単純文と複合文からなる．文の構成を Table 1 に示す．単純文は〔〔文名標〕文本体〕,. の形式で示される．ここに〔 〕はオプションを示す．

複合文は，その中に他の文を含む文であり，IF 文と ON 文からなる．

（2） グループ

DO グループは，つぎの形式で示される．

〔ラベル..〕DO 文
⋮
文
⋮
END〔ラベル〕,.

ここに DO グループの入れ子を許す．

（3） 手続きブロック

手続きブロックは，つぎの形式で示される．

入口名.. PROC 文
⋮
文
⋮
END〔入口名〕,.

ここに2次入口と入れ子は許すが，BEGIN ブロックは認めない．1番外側の手続きを外部手続き，手続きに含まれる手続きを内部手続きという．

（4） プログラム

プログラムは1つあるいは複数個の外部手続きからなる．

（5） データの宣言

データを使用する時には，DCL 文（宣言文），ラベル接頭語（入口名，文ラベル）により宣言することが必要である．手続きのパラメータは，パラメータ・リストに指定するこれにより宣言される．定数は使用することにより宣言されたと見なされる．宣言されたデータはその手続き，およびそれに含まれる手続きで同一の名標（Identifier）が宣言されない限り有効である．データ宣言の有効範囲として，外部手続きを越えて有効な EXTERNAL 属性と，手続き内でのみ有効な INTERNAL 属性がある．EXTERNAL データの宣言は，PL/I と相違し，外部手続きでのみ許す．内部手続きで EXTERNAL 属性の宣言を認めると外部手続き内の同一の EXTERNAL データに関して，宣言された手続きでのみ EXTERNAL データの宣言を有効としなければならないので，コンパイラの負担を増す．

2.3 記憶域管理

（1） 記憶域の種類

（a） プログラムのロード時からアンロード時まで引続き確保される記憶域（STATIC 記憶域）

（b） 手続きに制御が移行した時確保され，手続きから制御が抜け出す時返却される記憶域（AUTOMATIC 記憶域）

（c） 処理時，必要な時点で確保され，不要になった時点で返却される記憶域（BASED 記憶域）

（2） 割り付け方式

（a） STATIC データの割り付け方式

コンパイル時に目的プログラムの1部として領域を確保する．

（b） AUTOMATIC データの割り付け方式

この方式として，コンパイル時に STATIC データと同じように確保するか，実行時に AUTOMATIC データを宣言した手続きが呼出された時確保する方式がある．ここに前者は，STATIC データと同じ扱いとなり，AUTOMATIC データの存在意義を失わせる．後者は，実行時，手続きが呼び出された時に，この手続きのプロローグ処理として，AUTOMATICデータの領域を確保することになり実行時間のオーバヘッドを増す．そこで AUTOMATIC データの存在意義を持たせ，実行時のオーバヘッドをなくす方式とし

て，つぎの方式を採用する．

（i） AUTOMATIC データの領域をコンパイル時に割り付ける．

（ii） 同時に活きない（active にならない）手続きの AUTOMATIC データを，オーバレイさせて割り付ける．ここに active であるとは，ある手続きが呼び出された時点から，呼び出し元に復帰する時点までの間にある状態をいう．

（iii） AUTOMATIC データに INITIAL 属性の指定を禁止する．

INITIAL 属性の指定が許されると，INITIAL 属性を持つ AUTOMATIC データの宣言された手続きが，呼出される度（プロローグ時）に，この手続きの INITIAL 属性を持つ AUTOMATIC データに関する初期設定処理が必要になる．また AUTOMATIC変数の次元の上限，列の長さに定数以外の指定を禁止することにより，プロローグにおける次元の上限，列の長さの計算処理を不要とする．つぎに AUTOMATIC データの割り付け方式を例示する．

```
A.. PROC,.
  B.. PROC,.
      END B,.
  C.. PROC,.
      CALL E,.
      END C,.
  E.. PROC,.
      END E,.
      CALL B,.
      CALL C,.
    END A,.
```

このプログラムで手続き B と手続き C, E は，同時に active にはならない．それゆえ手続き B と手続き C, E の AUTOMATIC データをオーバレイさせて割り付ける．

（c） BASED データの割り付け方式

処理に必要な時点で，ALLOCATE 文により確保される．この時，BASED データの最後の要素の次元の上限，列の長さにだけ，定数以外の指定を認める．これにより，実行時のドープ・ベクトルの計算処理を不要とする．

2.4 割り込み管理

割り込み管理は，ON 文により行なう．

（1） ON 文の仕様はつぎの通りである．

ON 割り込み条件 ON ユニット

ここに（a）割り込み条件として，ENDFILE（ファイル名）と AREA 条件がある．（b）ON ユニットとしては，GO TO ラベル定数.. のみ許される．ここにラベル定数は，ON ユニットを有する手続きで定義される．（c）手続きに，割り込み条件を発生する可能性のある文がある時は，その文の実行前に，あらかじめ ON 文を指定する．さらにこのような手続き内に CALL 文がある時は，その直後に ON 文を指定する．（d）最近に発せられた ON 文が有効となる．（e）標準的な割り込み動作は認めない．

（2） 最近に発せられた ON 文を有効とする理由は，つぎの通りである．

つぎに示す例により評価する（本例は PL/I でかかれたプログラムとする）．

```
A.. PROC,.
    ON AREA ON ユニット4
      X5 文
    ON AREA ON ユニット1
    CALL B,.
      X1 文
    END A,.
  B.. PROC,.
/* ここで DATA 1, DATA 2 が使用される
*/
      ON AREA ON ユニット2
/* ON ユニット 2 内で，DATA 1, DATA 2,
DATA 3 が使用される */
/* ここで DATA 1, DATA 3 が使用される
*/
        CALL C,.
          X2 文
        END B,.
      C.. PROC,.
          X3 文
        ON AREA ON ユニット3
          X4 文
        END C,.
```

ここに手続き A, B, C の順序に制御の移行が行なわれるとする．

PL/I では，X5 文で AREA 条件が発生すると，ON ユニット 4 が実行される．X1 文で AREA 条件が発生すると，ON ユニット 1 が実行される．この機能を実現するためには，

(a) ON 文が出現したら，ON の文情報 (ON 種別，ON ユニットの番地) をプッシュ・ダウン (push down) する．

(b) 同一手続き内で同一の ON 条件に対する ON 文が出現すれば，後に出現した ON 文を有効とする．たとえば X_1 文で ON ユニット 4 が実行されず，ON ユニット 1 が実行される．

(c) ON 文を含む手続きが終了すれば，この手続きの有する ON 文の情報をポップ・アップ (pop up) する．この結果，この手続きを呼び出した手続きの ON 文が有効となる．たとえば X_1 文で ON ユニット 2 が実行されず，ON ユニット 1 が実行される．

このように ON 文の出現時点，あるいは ON 文を有する手続きの終了時点で，ON 文の情報のプッシュ・ダウン，ポップ・アップ処理が必要となる．このような処理を除外するために最近に発せられた ON 文を有効とする．このように規定すれば，ON 文の出現した時点で，ON 文の情報を更新 (overwrite) してゆくだけでよいことになる．この場合 CALL B,. と CAL LC,. の直後に ON 文を指定することが必要である．

(3) ON ユニットとして，GO TO ラベル定数,. だけしか許さない理由は，つぎの通りである．

PL/I で許される ON ユニットは，ラベルを持たない BEGIN ブロックか，単純文 (BEGIN, DO, END, RETURN, ENTRY, FORMAT, PROC, DCL 以外の文) である．

割り込みは，プログラムの正常な制御の流れに独立である．たとえば手続き C の X_3 文で AREA 条件が発生すると，手続き B の ON ユニット 2 が実行される．

レジスタ管理は，外部手続き単位に行なわれる．手続き B で，DATA 1, DATA 2, DATA 3 にレジスタ GR_i, GR_j, GR_k が固定的に割り付けられるとする．X_3 文で AREA 条件が発生すると，つぎのような処理が必要となる．

(a) GR_i, GR_j, GR_k を含む ON ユニット 2 のレジスタを復旧し，しかる後 ON ユニット 2 に制御を移行させなければならない．

(b) ON ユニット 2 から抜け出る時に，GR_i, GR_j, GR_k の内容を退避してやらなければならない．

このように (a), (b) の処理は，GR_i, GR_j, GR_k を固定した効果を低減させることになる．

これに対して，ON ユニットが GO TO ラベル定数,. の場合 (SYSL の ON 文の規定) には，ON ユニット内で使用される変数がないことから，ON ユニットに関するレジスタの復旧および退避処理が不要であること，ON 条件が発生した時には，無条件にラベル定数に制御を移行させればよいこと，しかもラベル定数は GO TO 文を含む手続き内で定義されており，手続きを異常終了させることはないので，手続きのエピローグ処理は不要であるなどの特性を有する．

このことより，PL/I で規定する ON 文の処理が実行時間のオーバヘッドを伴なうのに比べて，SYSL の ON 文の処理は実行時間のオーバヘッドが少ない．

(4) 標準的な割り込み動作を認めない理由は，つぎの通りである．

(a) 割り込み動作は，全てシステム・プログラマが管理できること．

(b) コンパイラが簡単になること．

2.5 追加された属性

(1) PLUS 属性

FIXED 属性の付加属性であり，2 進固定小数点変数が，値としてゼロか正の値だけ持つことを示す．たとえば，DCL A BIN (8),. の時，A は，符号ビットも含めて 16 ビットの領域がとられる．これに対して PLUS 属性が付加されれば，符号ビットが不要となり，8 ビットの領域で充分である．

(2) CONSTANT 属性

CONSTANT 属性は，変数の値がプログラム実行中変わらないことを示す．CONSTANT 属性は，レベル 1 の STATIC 変数 (AREA 属性を持つ変数は除く) に与えることができる．

(a) レベル 1 の CONSTANT 属性を持つスカラー・データが，演算式のオペランド，添字付名の添字あるいは SUBSTR 組込関数の第 2，第 3 アーギュメントに使用される場合，このデータは機械語命令のイミーディエイト・オペランド部，ディスプレースメント部あるいはオペランド長部に吸収され，メモリは取られない．

(b) CONSTANT 属性を持つ構造体と配列は，(a) と異なり，メモリ領域は取られるが，プログラム実行中変更されることがないので，タスク共有部とタスク固有部からなるリエントラント・プログラムの場合，タスク共有部にメモリを確保できる．したがって同時走行タスクが 2 以上の時，

メモリの節約になる.

3. 最適化技法

SYSL コンパイラが行なう主な最適化処理は，つぎの通りである．なお DIPS-1 の機械語命令については，文献 4) を参照のこと．

（1） DO ループ内の制御変数を添字として持つ変数の最適化.

（a） A(I+k) の番地＝A(1) の番地＋(I＋k－1)＊M_a （1）

＝A(1)の番地＋M_a＊k－M_a＋M_a＊I

＝$\alpha+\beta$＊I （2）

ここに M_a は A のマルチプライヤ，α は A(1) の番地＋M_a＊k－M_a，β は M_a.

（b） （a）で得られた $\alpha+\beta$＊I で α をディスプレースメントとベース・レジスタに入れ，β＊I をインデックス・レジスタに入れる．β＊I に関しては，まず β をインデックス・レジスタに設定し，DO ループが繰返すごとに β を加えれば，求める A(I) のアドレスが得られる．すなわち(1)式で示される加算，乗算を，(2)式で示される加算に変えることができ，実行時間が短縮される．**Fig. 1** に例を示す.

（2） DO ループに BXLE 命令を使う最適化.

DO 文の機能は，DO ループの繰返しの時点で，制御変数に増分を加え，この値を最終値と比較して最終値を越えなければ，つぎの繰返しを実行するものである．この処理を BXLE 命令のみで行なうことができ，複数個の命令で実行する場合に比べて，実行時間およびコア召有量が短縮される．**Fig. 2** に例を示す.

（3） AUTOMATIC なポインタ変数，外部手続きのパラメタ，EXTERNAL データの内，使用頻度の高いものにベース・レジスタを固定する最適化[5].

これらのデータの参照に関して，一般には参照時メモリ領域からのロード，参照後メモリ領域へのストア

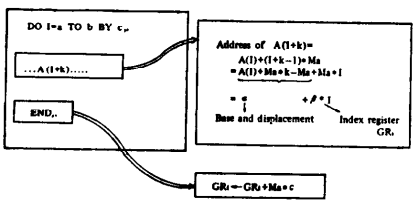

Fig. 1 Optimization of subscripted variables. which have control variables as subscripts, in DO loop

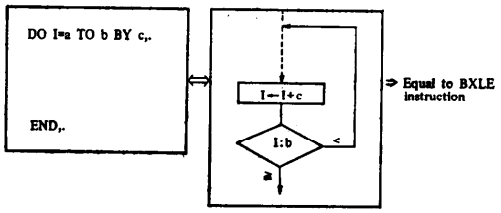

Fig. 2 Optimization of BXLE instruction used for DO loop control

が行なわれる．しかるにこれらのデータをベース・レジスタに固定すれば，手続きの入口でのみ，ロードを行えばよく，参照時のロード，ストアは不要となる．これにより実行時間およびコア召有量の短縮がはかられる.

（4） 使用頻度の高い AUTOMATIC な算術データ，BXLE 命令の使用するジェネラル・レジスタ群，DO ループの制御変数にジェネラル・レジスタを固定する最適化.

(3)と同じ方式により，実行時間およびメモリ召有量の短縮がはかられる.

（5） 共通式の再評価を行なわない最適化[5]

たとえば

A＝B＋C＊D,.

・

・

H＝I－C＊D,.

なるプログラムがある時，C＊D の演算を 2 回行なうのをやめて，1 回ですませる方式である．これにより時間，空間効率の向上がはかられる．**Fig. 3** に例を示す.

（6） A(I+k) 型添字付変数の番地計算に関する最適化.

A(I+k) 型添字付変数の番地計算は，つぎのようになる.

A(I+k) の番地＝A(1) の番地＋(I＋k－1)＊M_a （3）

ここに M_a は A のマルチプライヤ，k は定数である．(3)式を変形すると，(4)式をうる.

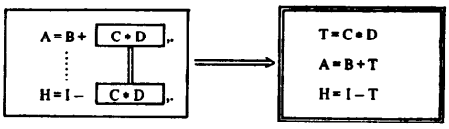

Fig. 3 Optimization for skipping re-evaluation of common expressions

A(I+k)の番地=A(1)の番地+(k−1)*M_a+I*M_a (4)

(4)式の (k−1)*M_a をコンパイル時に計算することにより，(3)式の I+k−1 の演算を実行時に行なう必要がなくなり，時間，空間効率の向上がはかられる．

(7) IF 文の THEN 節，ELSE 節がGO TO 文，RETURN 文である時，および飛び先が GO TO 文である時の最適化．

たとえば

IF A=B THEN GO TO L,.

の時，通常

CLV (比較) A,B
BF (偽なら分岐) GL
B (分岐) L
GL..

という中間言語が出力されるが，これに対して

CLV A,B
BT (真なら分岐) L

という中間言語を出力することにより，時間，空間効率の向上がはかられる．

(8) コンディション・コードの有効範囲内で，比較命令を削除する最適化．

たとえば，

IF A=B THEN ・・・ (5)
IF A>B THEN ・・・ (6)

という文が続いて出現した時，(5)文の中間言語で比較命令が発行されれば，(6)文の中間言語では，(5)文の結果を利用する．これにより時間，空間効率の向上がはかられる．**Fig. 4** に例を示す．

(9) ラベル間で，レジスタと変数の同値関係により，ロード命令を少なくする最適化[5]．

ラベルとラベルの間に制御が移行することはない．したがって，1度ある変数にレジスタが対応づけられると，そのレジスタの内容をそのラベル間で破壊しない限り，変数とレジスタの対応関係は保持される．これを利用してロード命令を少なくし，最適化を行なう

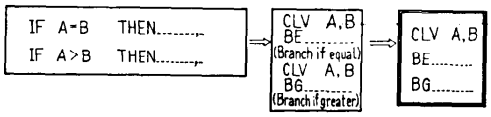

Fig. 4 Optimization for deleing "Compare" instruction in the scope of condition codes

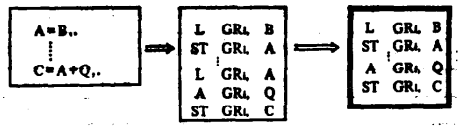

Fig. 5 Optimization for suppressing redundant "load" instructions by fixing registers to variables between labels

ものである．**Fig. 5** に例を示す．

(10) 使用頻度の高い組込関数，列操作などにイン・ライン・コードを出力し，実行時間を短縮する最適化．

4. 評 価

SYSL コンパイラの完成後，評価を行なった．評価に使用したサンプル・プログラムの特性を **Table 2** に，サンプル・プログラムの大きさ，ならびに生産性（コーディング期間，デバグ期間およびバグ件数で示す）を **Table 3** に，目的プログラムの実行時間ならびにコア召有量を，それぞれ **Table 4**, **Table 5** (4, 5 は次頁参照) に示す．Table 3, 4, 5 は．それぞれ SYSL 言語とアセンブラ言語との比較で示されている．

Table 2 Sample programs charactristics

Coding Technique / Programs	Table Handling	Character String Processing	Bit String Processing	List Processing	Other Processing
P_1	20 (%)	40 (%)	10 (%)	10 (%)	20 (%)
P_2	50	10	20	0	20
P_3	30	40	10	10	10
P_4	20	60	0	10	10
P_5	30	60	0	0	10
P_6	10	80	0	0	10
P_7	10	20	40	10	20
P_8	0	50	0	40	10
P_9	30	40	0	20	10
P_{10}	10	40	0	20	30

Table 3 Productivity and number of sample programs source statements

Program	Coding Period (day)		Debugging Period (day)		Number of Bugs		Number of Source Statements	
	SYSL	Assembler	SYSL	Assembler	SYSL	Assembler	SYSL	Assembler
P_1	2	3.5	4	6	5	12	495	668
P_2	1	5	4	8	5	6	109	164
P_3	4	8	5	12	23	20	558	485
P_4	3	2	3	5	7	10	300	409
P_5	1	1	2	1	3	3	79	145
P_6	1.5	2	4	5	6	5	119	226
P_7	1.5	2	3	10	2	23	135	175
P_8	1.5	2	3	5	2	6	89	149
P_9	2	4	5	7	7	12	343	511
P_{10}	1.5	2	3	6	6	10	203	346
Gross	19.0	31.5	36	70	66	107	2430	3278

Table 4 Run time comparison

Programs	SYSL	Assembler	SYSL/Assembler
P_1	20579 (m)	17333 (m)	1.19
P_2	9514	7523	1.26
P_3	5350	5346	1.00
P_4	16743	15743	1.06
P_5	1548	2007	0.77
P_6	3576	2538	1.41
P_7	12630	12436	1.02
P_8	1418	1582	0.90
P_9	2453	1765	1.39
P_{10}	1425	1264	1.13
Gross	75236	67518	1.11

Table 5 Memory space comparison

(bytes)

Language Category / Programs	SYSL			Assembler		
	Data Part	Instruction Part	Gross	Data Part	Instruction Part	Gross
P_1	10127	2640	12767	9807	1938	11745
P_2	288	804	1092	636	832	1468
P_3	1708	2112	3820	1952	1352	3304
P_4	1624	1536	3160	1100	1392	2492
P_5	600	756	1356	280	576	856
P_6	316	764	1080	184	688	872
P_7	772	876	1648	608	636	1244
P_8	767	592	1359	645	488	1133
P_9	1896	1984	3880	760	1688	2448
P_{10}	752	1160	1912	684	1056	1740
Gross	18850	13224	32074	16656	10646	27302

Data part: $\frac{\text{SYSL}}{\text{Assembler}} \fallingdotseq 1.13$

Instruction part: $\frac{\text{SYSL}}{\text{Assembler}} \fallingdotseq 1.24$

Gross: $\frac{\text{SYSL}}{\text{Assembler}} \fallingdotseq 1.17$

5. あとがき

SYSL 開発により.

(1) SYSL の生産性が，アセンブリ言語の生産性に比較して，コーディング期間，デバグ期間，バグ数の観点より，約2倍程度であること.

(2) SYSL の目的プログラムは，アセンブリ言語の目的プログラムに比較して，実行効率，メモリ効率の点で，それぞれ 1.11 倍，1.17 倍であること.

(3) SYS-L により，DIPS-1 OS の処理プログラムの 90% 以上を記述できること.

などが明らかとなり，システム・プログラム製造に充分使用できることが確認できた．今後の問題点としては，メモリ効率を改善するために,

(1) コンパイラが生成したデータのオーバレイ.

(2) ユーザ定義データの内，参照のないものの削除.

(3) PROC-ENTRY, ENTRY-ENTRY と文が連続している時，1文に集約する.

などがある.

実行効率を改善するために,

(1) MIN, MAX 組込関数の導入などがある.

現在，SYSL の適用領域を拡張し，制御プログラムを含む OS 全般を記述できるように検討中である.

おわりに，本研究を推進するにあたり，御指導いただいた横須賀電気通信研究所 戸田調査役に深甚の謝意を表します．なお本研究は，日本電気(株)，(株)日立製作所，富士通(株)との共同研究として行なわれたものである.

参 考 文 献

1) Takashima, K et al.: A Large-scale Data Processing System : DIPS-1, First USA-Japan Computer Conference Procedings. pp. 193～202 (Oct., 1972).
2) 関口良雅，高原靖，岸上利秋：DIPS の実用化標準化された電電公社の電子計算機，電子通信学会誌 57, 10, pp. 1139～1159 (Oct., 1974).
3) IBM System/360 Operating System : PL/I Language Specifications, C 28-6571-4, IBM Corp. (1966).
4) 山田他：DIPS-1 論理装置，通研実報 21, 10, pp. 1837～1877 (Oct., 1972).
5) Lowry, E. S. and Medlock, C. W : Object Code Optimization, Comm. ACM 12, 1, pp. 12～22 (1969).

(昭和 49 年 9 月 24 日受付)
(昭和 49 年 11 月 5 日再受付)