資　料

システム製造用言語 SYSL-2 の設計*

寺島信義** 高橋宗雄**

Abstract

SYSL-2, System Description Language, is a system implementation language for the use by DIPS-1 operating system, especially for executive programs.

SYSL-2 has been extended to cover a wider scope than SYSL. This paper deals with the concept of the operating system description and the summary of SYSL-2 language specifications.

1. まえがき

DIPS-1 用高能率システム製造用言語として，SYSL[1]を実用化したが，これは，主としてOSの内，処理プログラムの記述・製造を行なうものであった．われわれは，SYSL の実績に基づいて，OS の内，特に制御プログラムの記述・製造を目的として，SYSL-2 を実用化した．SYSL-2 は SYSL を包含している．本論文では，**2.** で OS 記述に必要な基本概念を述べ，**3.** で SYSL-2 の言語概要について，SYSL との相違点を中心に述べる．

2. OS 記述に必要な基本概念

OS は，文献 2) に基づき，論理的に

（1） 記憶域
（2） データ
（3） モジュール

からなると考える．

これらの関係には，つぎの3通りがある．

（1） 記憶域～データあるいはモジュール　割り付け関係
（2） データ～モジュール　アクセス関係
（3） モジュール～モジュール　呼び出し関係

2.1 割り付け関係

プログラムは，データとモジュールからなるとみなす．モジュールには，システム生成時に割り付けられ，システムが消滅するまで存在するものと，モジュールが使用される時に記憶域を割り付けられ，使用後開放されるものに分類される．これらは，それぞれ PL/I[3] の STATIC, AUTOMATIC 記憶域に対応する．データには，上記のほかに，処理に必要な時点で割り付けられ，不要になった時点で開放されるものがある．これは，PL/I の BASED 記憶域に対応する．

2.2 アクセス関係

データに対するアクセスを考える場合には，データの種類が明らかでなければならない．

データの種類としては，ビット列，文字列，算術(2進，固定/浮動小数点)，ラベル，入口，ロケータ，領域データがある．これらは，それぞれ PL/I の BIT, CHAR, BIN FIXED/FLOAT, ENTRY, POINTER/OFFSET, AREA データに対応する．これらに対するアクセスが可能であることが必要である．これらの仕様は，基本的に PL/I で提供される仕様のサブセットで十分である．

2.3 呼び出し関係

モジュール～モジュールのインタフェース要因にはつぎの3つがある．

（1） レジスタ・コンベンション
（2） リンケージ・コンベンション
（3） モジュールで使用する作業域のコンベンション

DIPS-1 のレジスタの個数は，有限（汎用レジスタ(GR)）；16個，ベース・レジスタ(BR)；16 個であるのでモジュール(i)からモジュール(j)を呼び出し，モ

---

* SYSL-2-System Description Language by Nobuyoshi TERASHIMA and Muneo TAKAHASHI (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

ジュール(j)からモジュール(k)を呼び出すという風につぎつぎとモジュールの呼び出しが起る時，各モジュールで必要なだけレジスタを確保してゆけば，レジスタが不足してしまう．このような事態を回避するために，モジュール間でレジスタの取り決めをしておく必要がある．この取り決めを**レジスタ・コンベンション**と呼ぶ．

レジスタ・コンベンションとして，つぎのような方法が考えられる．

(1) その手続きで使用するレジスタをコンパイラが認識し，それらを退避・回復する方法．

(2) システムで保証すべきレジスタを一意に決め，決められたレジスタの退避・回復をコンパイラが行なう方法．

(3) プログラマに退避・回復すべきレジスタを指定させ，それらをコンパイラが退避・回復する方法．

(1)はコンパイラ側で退避・回復処理を行なう方式である．通常レジスタの割り付けが行なわれると，目的プログラムで使用されるレジスタが決定される．これに基づいてコンパイラは退避・回復用の命令を出力することになる．

(1)の処理を簡略化するために，退避・回復処理の対象となるレジスタを一意に決める方法がある．これが(2)の方法である．

この方法では，退避・回復すべきレジスタが決まっていることからレジスタ割り付け処理と，退避・回復処理を同時に行なうことができる．この時退避・回復処理の対象とならないレジスタが使用される時には，他の手続きに制御が移行すると，一般にこれらのレジスタは保証されないので，他の手続きに制御移行が行なわれるたびに，これらのレジスタの退避・回復処理を行なう必要がある．この時，手続きによっては，退避・回復処理の対象となるレジスタの一部しか使用されない場合がある．このような場合は，不必要に退避・回復処理が行なわれることになる．このような冗長さを除外するために，プログラマに退避・回復の対象となるレジスタを指定させる方法がある．これが(3)の方法である．

この方法では，プログラマはコンパイラがその手続きの目的プログラム生成のために，どれだけレジスタを使用するのか予測することが必要である．コンパイラのレジスタ割り付けアルゴリズムをプログラマが周知していれば，プログラマがこの予測を行なうことは困難ではない．このとき，予測に反して退避・回復処理の対象とならないレジスタが使用される時には，(2)の方法と同様に，他の手続きに制御移行が行なわれる度に，これらのレジスタの退避・回復処理が必要となる．

また SYSL-2 プログラムに CODE ブロック (**3.3 (2)**参照) を記述することができる．このブロック内で機械語を使用することができる．機械語では直接レジスタ番号が使用されたり，間接的に使用されたりする．これをコンパイラが認識するのは大変である．かりに CODE ブロックの中で使用されるレジスタが認識できたとすれば，(1)の方法では，それらも退避・回復の対象としなければならない．(2)の方法では，これらのレジスタの内，一意に退避・回復の対象と決めたレジスタに含まれないものについては，他の手続きに制御移行が行なわれる度に，退避・回復処理を行なう必要がある．これに対してプログラマは機械語で使用されるレジスタを認識している．また上述したようにプログラマはコンパイラが目的プログラム生成のために，どれだけレジスタを使用するかを予測することもできる．

以上の検討より各方式の比較評価を行なうとつぎのようになる．(2)と(3)の方法では，プログラマの予測が正しければ，(3)の方が，一般には目的プログラムの効率はよくなる．(1)と(3)の方法では，(3)にはプログラマの予測が入ることから，(1)の方が正確な処理を行なうことができる．しかしその反面コンパイラの処理が複雑となり，したがってコンパイル効率の低下などの問題を生ずる．それゆえ SYSL-2 では，(3)の方法をとることとし，(1)の方法は今後の研究課題とすることとした．(3)の方法はいわば，コンパイラとプログラマが協力して，目的プログラムの最適化を行なう方法ということができよう．

リンケージ・コンベンションは，モジュールからモジュールへの制御移行に関するものであり，

(1) パラメタの授受方法

(2) 制御移行方法

に分類できる．

パラメタの授受方法には，パラメタのアドレスのリストからなるパラメタ・リストのアドレスをレジスタ経由で渡すやり方，パラメタのアドレス（この場合，パラメタは1個に限られる）をレジスタ経由で渡すやり方や，パラメタをレジスタに設定して渡すやり方などがある．

制御移行方法は，リング・レベル（モジュールのプ

ロテクションを行なうために，モジュールを階層化する手段)，リンク契機(静的リンク，動的リンク)などにより異なる．

モジュールで使用する**作業域のコンベンション**として，つぎの2通りのやり方がある．

(1) caller 側のモジュールで作業域を用意し，これを called 側で使用するやり方 (CALL DATA コンベンションという)．

(2) 自分の必要とする作業域を自分のプログラムで，種々の方法により確保するやり方．

種々の方法として，CSECT 域（タスク共用域）あるいは PSECT 域（タスク固有域）に確保する方法や OS マクロを使用してモジュールの入口で確保する方法がある．

3. 言語仕様概要

SYSL-2 は，SYSL に，OS 特に制御プログラム記述に必要な機能を追加したものである．したがって，SYSL-2 の文法則，意味則は，基本的に PL/I と同じである．本論文では，SYSL-2 と SYSL の相違する部分について，主たる項目を述べる．

3.1 データの種類と構成

BINARY FLOAT データ，ENTRY データが追加されている．16 進定数は，BIT 列定数の簡便な表記法として使用される．データ構成としては，3次元までの配列および構造体の配列が許される．

3.2 プログラム構造

PL/I の BEGIN ブロックが導入される．

3.3 その他の特徴的仕様

(1) コンベンションについて

(a) レジスタ・コンベンションについて

ベース・レジスタ (BR)，汎用レジスタ (GR) の内，退避・回復の対象となる，レジスタ群を PROC 文の OPTIONS オプションで指定すると同時に，このモジュールを呼び出すモジュールで，このモジュールに関する宣言を行なう所で指定する．

たとえば

A: PROC OPTIONS (BR(I-J) GR (K-L)); と指定した時，退避・回復の対象となるレジスタは，BR について I から J まで，GR について K から L までであり，退避・回復処理をコンパイラが行なうことを指定する．

上記に示した指定と同時に，モジュール A を呼び出すモジュールで，DCL A ENTRY OPTIONS (BR (I-J) GR (K-L)); と指定する．

(b) 作業域コンベンションについて

このモジュールで使用する作業域を，タスク共用域 (CSECT)，あるいはタスク固有域 (PSECT) に確保するか，caller 側で確保した作業域を使用するかを，PROC 文の OPTIONS オプションで指定すると同時に，このモジュールを呼び出すモジュールで，このモジュールに関する宣言を行なう所で指定する．

たとえば

A: PROC OPTIONS (MEMORY (PSECT)); のように指定すると同時に，モジュール A を呼び出すモジュールで

DCL A ENTRY OPTIONS (MEMORY (PSECT)); と指定する．

(c) リンケージ・コンベンションについて

called 側モジュールに関する種々の特性（レジスタ・コンベンション，作業域コンベンション，パラメタ授受方法，リンケージの種別等）を，このモジュールに関する ENTRY 属性の OPTIONS オプションで指定すると同時に，このモジュールの PROC 文の OPTIONS オプションで指定する．

たとえば

DCL B ENTRY OPTIONS (LINK MEMORY (CSECT) BR (I-J) GR (K-L)); と指定した時，入口名 B はダイナミック・リンクの対象となり (LINK オプション)，モジュール B のデータは CSECT にとられ，パラメタ・リストのアドレスが渡され，退避・回復の対象となるレジスタは BR が I から J，GR が K から L であり，これらのレジスタをコンパイラが退避・回復することを示す．

(2) CODE ブロック[4]の導入

OS の内，制御プログラムは機械とのインタフェースが強いこと，入出力部分の取扱いが必要なことなどから，これらを SYSL-2 に取り込むと SYSL-2 が機械（システム）依存になってしまう．これを避けるために CODE ブロックを導入し，ここで機械依存部分を記述する．CODE ブロックは，SYSL-2 の単一の文が書ける所にはどこにでも記述できる．

データ宣言の有効範囲に関して，CODE ブロックは SYSL-2 の単一文の効果と同じである．すなわち CODE ブロックを含む BEGIN ブロック，あるいは PROC ブロックで宣言され，CODE ブロックで有効な変数を CODE ブロックで参照できる．

CODE ブロックでは，DCL 文によるレジスタ参照

変数の宣言，DC 命令 (IBM S/360 アセンブリ言語の DC (Define Constant) 命令に同じ) による文字列，ビット列，算術定数の定義が可能であるが，これらの宣言はその CODE ブロックでのみ有効である．CODE ブロックでは，アセンブリ言語に似た機械語 (同族命令 (後述(a)参照) か & 記号で始まる機械語で，たとえば〈L〉や & MVM など) を記述することができる．機械語のオペランドとしては，BEGIN ブロックあるいは PROC ブロックで宣言され CODE ブロックで有効な変数，レジスタ番号，レジスタ参照変数，DC命令で定義される定数や直接アドレス指定 (d (r_1, r_2)；ここに d はディスプレースメント，r_1 はインデックス・レジスタ (汎用レジスタ)，r_2 はベース・レジスタである) が指定できる．このほか記憶域確保・開放文，FORGET 文 (後述 (b) 参照)，レジスタ確保・開放文 (後述(c)参照) が使用できる．しかし，CODE ブロックのある SYSL-2 プログラムは，結果的に機械依存になってしまう．これを避けるために，CODE ブロックを SYSL-2 で提供されるマクロで記述することにより，CODE ブロックを SYSL-2 プログラムの外側においやる．

つぎに CODE ブロックの使用例を示す．

```
CODE;
DCL GRX GR;
REGREQ GRX;
〈L〉 GRX, LENG;
& BCTR GRX, 0;
& TR STR, TBL, GRX;
REGFRE GRX;
ENDCODE;
```

この例はつぎのような意味である．

CODE ブロックで宣言された，レジスタ参照変数 GRX に対して，REGREQ 文 (後述(d)参照) で，コンパイラにレジスタを要求する．得られたレジスタを使用して，TR (Translate) 命令により，STR 変数をテーブル TBL のパターンに従がって，LENG 分だけ変換を行なうものである．ここに LENG, STR, TBL は，この CODE ブロックを含む PROC ブロックあるいは BEGIN ブロックであらかじめ宣言された変数である．〈L〉は同族命令であり，LENG の値がレジスタ (GRi) 上にあれば LR GRX, GRi となり，メモリ上にあれば L GRX, LENG となる．REGFRE 文は GRX に割り付けられたレジスタを開放する．

(a) 同族命令の導入

同族命令とは，本質的には同じ機能であるが，そのオペランドの所在 (レジスタ上かメモリ上か) により異なる機械語命令コードの総称である．たとえばロード動作に対して，DIPS-1 では L, LR, LC, LH などの機械語命令が存在するが，これを〈L〉で総称する．通常コンパイラは，SYSL-2 プログラム内の BEGIN ブロックや PROC ブロックで宣言された変数や式の値がメモリ上にあるかレジスタ上にあるかコンパイル時に追跡している．たとえばこのような変数にアクセスする際に，その変数の値がレジスタ上にあればその値を使用した方が目的プログラムの効率がよくなる．

しかるに CODE ブロックを記述するプログラマには，このような変数がレジスタ上にあるのかメモリ上にあるのかわからない．この場合同族命令を用いればコンパイラはオペランドの所在を調べ，それにより最適な機械語を選択する．同族命令は，〈同族命令コード〉命令のオペランド群；で表現される．

(b) FORGET 文

機械語命令のオペランドに，直接アドレスが指定された時，コンパイラは，そのアドレスに対応する変数 (その機械語命令のある CODE ブロックを含む PROC ブロックあるいは BEGIN ブロックで宣言され，CODE ブロックで有効な変数) が何かわからない．その変数の値がレジスタ上のみにある時，その機械語が実行されると，変数の値とその変数に対応するレジスタ上の値がくい違ってくる．このようなくい違いを除去するために，プログラマは，FORGET 文を指定することが必要である．FORGET 文が指定されると，コンパイラは，レジスタ上にある変数の値をメモリ上に退避し，その変数に割り付けられていたレジスタを開放する．つぎに例を示す．

```
DCL (A, B) BIN FIXED;
      :
CODE;
DCL BRI BR;
REGREQ BRI;
& MVM 0 (0, BRI), A, 4;
      :
ENDCODE;
```

ここで，& MVM 命令で，0 (0, BRI) で示されるアドレスに，A の内容が 4 バイト分転送される．0 (0, BRI) が，もし変数 B だとしても，コンパイラには一般にわからない．この時Bの値がレジスタ上に

引き上げられている場合に，& MVM 命令を実行した後の B の値と B に対応するレジスタの値の不一致が起きる．これを避けるために，& MVM 命令の前に FORGET B; を指定することが必要である．

（c） レジスタ割り付けおよび開放区間 (segment) の指定

CODE ブロック内の機械語で使用するレジスタの指定法には，レジスタ参照変数による方法と定数による方法がある．定数による指定は，OS の取り決めにより，特定レジスタを使用しなければならない場合に使用する．レジスタ参照変数による指定は，任意のレジスタでよい場合に使用し，コンパイラにレジスタの割り付けを依頼する．この時レジスタの有効利用をはかるために，CODE ブロック全体でレジスタ参照変数にレジスタを割り付けずに，CODE ブロックの必要区間 (segment) で割り付ける．これを REGREQ 文，REGFRE 文により行なう．

（3） その他の機能

（a） REGISTER 属性

SYSL コンパイラでは，使用頻度の高い変数に，レジスタを手続き内で固定的に割り付けたり，DO ループの制御変数を，レジスタにループ内で固定的に割り付けるなどの最適化を行なった．前者は手続き内の静的な使用頻度により割り付けるので，必ずしも動的な使用頻度とは一致しない．この欠点を除去するために SYSL-2 では REGISTER 属性を導入し，プログラマに使用頻度の高い変数を，レジスタに固定的に割り付けるよう指定させる．さらにレジスタ割り付けを，よりきめ細かに管理するために，レジスタに固定して割り付ける区間を指定することもできる．(RESTRICT 文，RELEASE 文による) これは，PL/S[5] と同じ考え方である．

（b） ポインタ演算

SYSL では，ポインタ演算が導入されなかったが，OS 記述ではポインタ演算の使用頻度が高いことが判明したので，SYSL-2 ではつぎに示す形式のポインタ演算が導入される．

スカラー・ポインタ変数＝スカラー・ポインタ変数 ±{整数値を返すスカラー算術式}；

（c） INDEX 属性

配列要素 A(I) の参照がある時，A(I) のアドレスは，次のように演算される．

Address of A(I) = Address of A(1) + M_a * (I−1)

ここに M_a は配列 A の 1 要素の大きさで，配列 A のマルチプライヤと呼ばれる．

つまり A(I) の参照があるたびに，上記のアドレス計算が行なわれる．この時上式の第 2 項の掛算を除外するために INDEX 属性が導入される．

つぎに例を示す．

つぎに示すようなプログラムがあるとする

```
I=2;・・・(1)
  :
C=A(I)+B;・・・(2)
  :
D=A(I)+D;・・・(3)
```

この時，I が INDEX 属性を持つ場合（ケース 1）と I が INDEX 属性を持たない場合（ケース 2）でつぎのような処理が行なわれる．

（ケース 1） I が INDEX 属性を持つ時．

（1）文に対して；I に M_a*2 の演算結果が代入される．

（2）文に対して；A(I) のアドレスは，A(1) のアドレスに (I-M_a) の値を加えたものとなる．

（3）文に対して；上記（2）文の処理と同じ処理が行なわれる．

（ケース 2） I が INDEX 属性を持たない時．

（1）文に対して；I に 2 が代入される．

（2）文に対して；A(I) のアドレスは，A(1) のアドレスに M_a*(I-1) の演算結果を加えたものとなる．

（3）文に対して；上記（2）文の処理と同じ処理が行なわれる．（ケース 1）では M_a に関する掛算が 1 回行なわれるのに対して，（ケース 2）では M_a に関する掛算が配列要素 A(I) の出現回数（2 回）分行なわれる．

この例からも明らかなように，（ケース 1）では INDEX 属性を持つ添字 I が代入文の左辺に現われる時のみ M_a の掛算が行なわれ，配列要素 A(I) の参照時には M_a の掛算は行なわれないのに対して，（ケース 2）では添字 I が代入文の左辺に現われる時は，M_a の掛算が行なわれず，配列要素 A(I) の参照時に M_a の掛算が行なわれる．さて添字が代入文の左辺に現われる回数 m 回，その添字を持つ配列要素が使用される回数 n 回とすると，一般には $m \leq n$ が成り立つ．したがってマルチプライヤの掛算回数は（ケース 1）≤（ケース 2）となる．この関係から配列要素の参照回数が多ければ多い程，INDEX 属性の効果は顕著となる．

（d） UNSPEC 擬変数

ある変数 (BIT 属性以外の) が使用される時，シス

テム・プログラムではBIT列として取扱かうことが多い．この場合SYSLではBASED変数で扱かっていたが，記述性および効率向上のために，SYSL-2ではUNSPEC擬変数が導入される．仕様はPL/Iと同じ．

つぎに例を示す．

4バイトの算術データCに，32ビットのビット列データBを，そのままのイメージ（変換せずに）で代入する時，SYSLではつぎのように記述される．

```
DCL P PTR, BB BIT (32) BASED;
  P=ADDR(C);
  P -> BB=B;
```

ここにADDR組込関数は，算術データCのアドレスを得る機能を持つ．

これに対してSYSL-2ではつぎのように記述される．

```
UNSPEC(C)=B;
```

ここにUNSPEC擬変数は，アーギュメントCをビット列とみなして，右辺Bのビット列をそのまま変換せずにCの領域に代入する機能を持つ．

これらに対する目的プログラムは，つぎのようになる．

SYSLプログラムに対して；

```
LBA  BRi, C
ST   BRi, P
L    GRj, B
ST   GRj, 0(, BRi)
```

SYSL-2プログラムに対して

```
L   GRj, B
ST  GRj, C
```

この例からも明らかなように，UNSPEC擬変数は記述性，および目的プログラムの効率向上に有効である．

（e） 59文字セット

SYSLでは49文字セットであったが，記述性向上のために，SYSL-2ではこれに加えて59文字セットを導入する．

4. おわりに

SYSL-2コンパイラは既に作成され，DIPS-1 OSの記述・製造に使用されている．

SYSL-2では，制御プログラム記述のために

（1） 機械（システム）依存部分の記述機能の導入
- CODEブロック

（2） レジスタ管理を，プログラマとコンパイラが協力して行なう手段の導入
- レジスタ変数およびレジスタ参照変数
- RESTRICT文，RELEASE文，REGREQ文およびREGFRE文
- 同族命令およびFORGET文

などを行なった．

SYSL-2はDIPS-1のOS記述が目的であるところから，DIPS-1 OS依存になった面がないとはいえない．今後DIPSに限らずOS一般の記述言語のあり方を，SYSL-2の運用経験データも積み上げながら検討してゆきたい．

最後に，日頃御指導いただく横須賀電気通信研究所戸田調査役および筑後室長に感謝いたします．

なお，本研究はDIPS-1プロジェクトの一環として横須賀電気通信研究所および日本電気(株)の共同研究として行なわれたものである．

ここに関係各位に感謝します．

参考文献

1) 寺島：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, pp. 492～498 (1975)
2) 林：システム設計言語DEAPLANについて，情報処理，Vol. 14, No. 9, pp. 652～660 (1973)
3) IBM System/360 Operating System : PL/I Language Specifications, IBM Corp. (1966)
4) Bergeron, R. D. et al.: Language for System Development, SIGPLAN Notices Vol. 6, No. 9, pp. 50～72 (1971)
5) Guide to PL/S-Generated Listings, IBM Crop. (1972)

（昭和49年12月17日受付）
（昭和50年2月5日再受付）