B2 PL/1WによるTSS用コンパイラコンパイラ作成

二村良彦，西野秀毅，吉村一馬（日立中央研究所）

要旨

HITAC 5020 TSS用に作成したコンパイラコンパイラVITALの概要を述べる．作成に用いた言語はPL/1Wである．コンパイラコンパイラは与えられたシンタクスとセマンティクスの記述に対してPL/1Wで書かれたコンパイラを作り出す．このコンパイラは与えられたソースプログラムに対してHITAC 5020用機械コードを直接作り出す．

1. 緒言

Mondshein[1]によれば，「VITALシステムの目的は，コンパイラ作成の際のこまかい仕事を自動的に行なうことである．VITALという名前は初期状態を変えることのできる算法言語のトランスレータ（Variably Initialized Translator of Algorithmic Languages）を意味する．システムの構成は図1のとおりである．

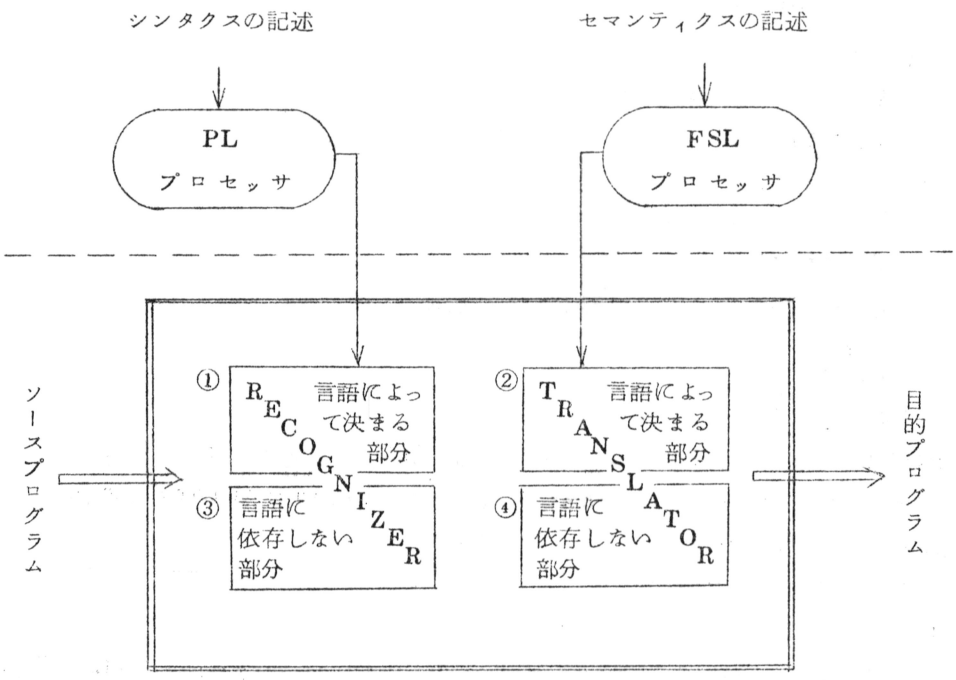

図1：VITALシステムの構成

コンパイラコンパイラは，PLプロセッサ，FSLプロセッサ，③，及び④から構成される．ソース言語のシンタクスとセマンティクスをPLプロセッサとFSLプロセッサが処理して，各各①と②を作る．①と③が一緒になってrecognizer（識別装置）を作り，②と④が一緒になってtranslator（翻訳装置）を作る．太枠で囲まれたブロックが生成されたコンパイラである．

本文はHITAC 5020 TSS用に作成されたVITALシステムの構成方法を述べるものである．システムの作成はPL/1 W[2]を用いて行なわれた．

2. PLプロセッサとFSLプロセッサ

PLプロセッサは，プロダクション言語（PL）で記述されたソース言語（定義しようとしている言語）のシンタクスに基づいてソース言語のrecognizerを作る．recognizerはソースプログラムの構文を識別しその構文に対応するセマンティックルーチンを呼出す．セマンティックルーチンの集合が図1におけるtranslatorを構成する．ここでは説明の都合上recognizerはソースプログラムを木構造に変換すると考える．

我々のシステムにおいてPLプロセッサ（以後Pl_pと略記することもある）は，PL/1 Wで書かれたrecognizerを作り出す．即ち，Sklansky等[3]の公式を用いれば，

$$Pl_p = \frac{Pl}{Pl_1 \quad Pl_1}$$ 註1)

である．ただし

Pl ：プロダクション言語

Pl_1：PL/1 W

を表わすものとする．

L：ソース言語

T：木構造の集合

とおけば，ソース言語シンタクスのPlによる記述は

$$\frac{L}{Pl \quad T}$$

と考えられる．

FSLプロセッサは，Formal Semantic言語（FSL）で記述されたソース言語のセマンティクスに基づいて図1におけるtranslatorを作る．translatorは，recognizerの作った木構造に作用して目的コードを作る．

註1) L_1，L_2，及びL_3が各々プログラミング言語を表わすとき，

$\frac{L_1}{L_3 \quad L_2}$ はL_1のプログラムをL_2のプログラムに翻訳し，しかもそれ自身はL_3で書かれているプロセッサを意味する．

我々のシステムにおいてFSLプロセッサ（以後Fsl_pと略記することもある）は，PL／1Wで書かれたtranslatorを作る．translatorの目的コードは5020 TSSの機械語である．即ち，

$$Fsl_p = \frac{Fsl}{Pl_1 \quad Pl_1}$$

ソース言語セマンティクスのFSLによる記述は，

$$\frac{T}{Fsl \quad M}$$

である．ただし，

Fsl：Formal Semantic言語

M　：5020 TSS用機械語

を表わすものとする．

PL／1WコンパイラをPl_{1c}で表わせば，

$$Pl_{1c} = \frac{Pl_1}{M \quad M}$$

である．ソースプログラムをq／L[註2)]，入力データをδで表わせば，我々のシステムにおける各プロセッサの相互関係は図2[註3)]で示される．図2においてメタコンパイルタイムと名付けられた（2本の破線にはさまれた）部分は，ソース言語の文法を読込んでrecognizerとtranslatorを作るフェイズを示し，コンパイルタイムはソースプログラムを目的プログラムに翻訳するフェイズを示し，そしてランタイムは目的プログラムが走るフェイズを示している．この用語法はFeldman[4)]に従った．

言語設計者がプログラミング言語，例えばL，をPLとFSLで記述した場合に3つのフェイズがどのように働くかを説明する．

註2）q／Lはプログラミング言語Lで書かれたアルゴリズムqを意味する．

註3）

は，データ（又はプログラム）Dをプログラム言語Mで書かれたプログラム（又はプロセッサ）Fで処理した結果がD′であることを意味する．これを，D′＝MFDで表わす．iはプロセスに付けられた番号である．

例　$F = \frac{Pl_1}{M \quad M}$，　$D = \frac{Pl}{Pl_1 \quad Pl_1}$　とするとき，

$MFD = M \frac{Pl_1}{M \quad M} \frac{Pl}{Pl_1 \quad Pl_1} = \frac{Pl}{M \quad Pl_1}$　なる算術ができる．詳細は文献3）参照．

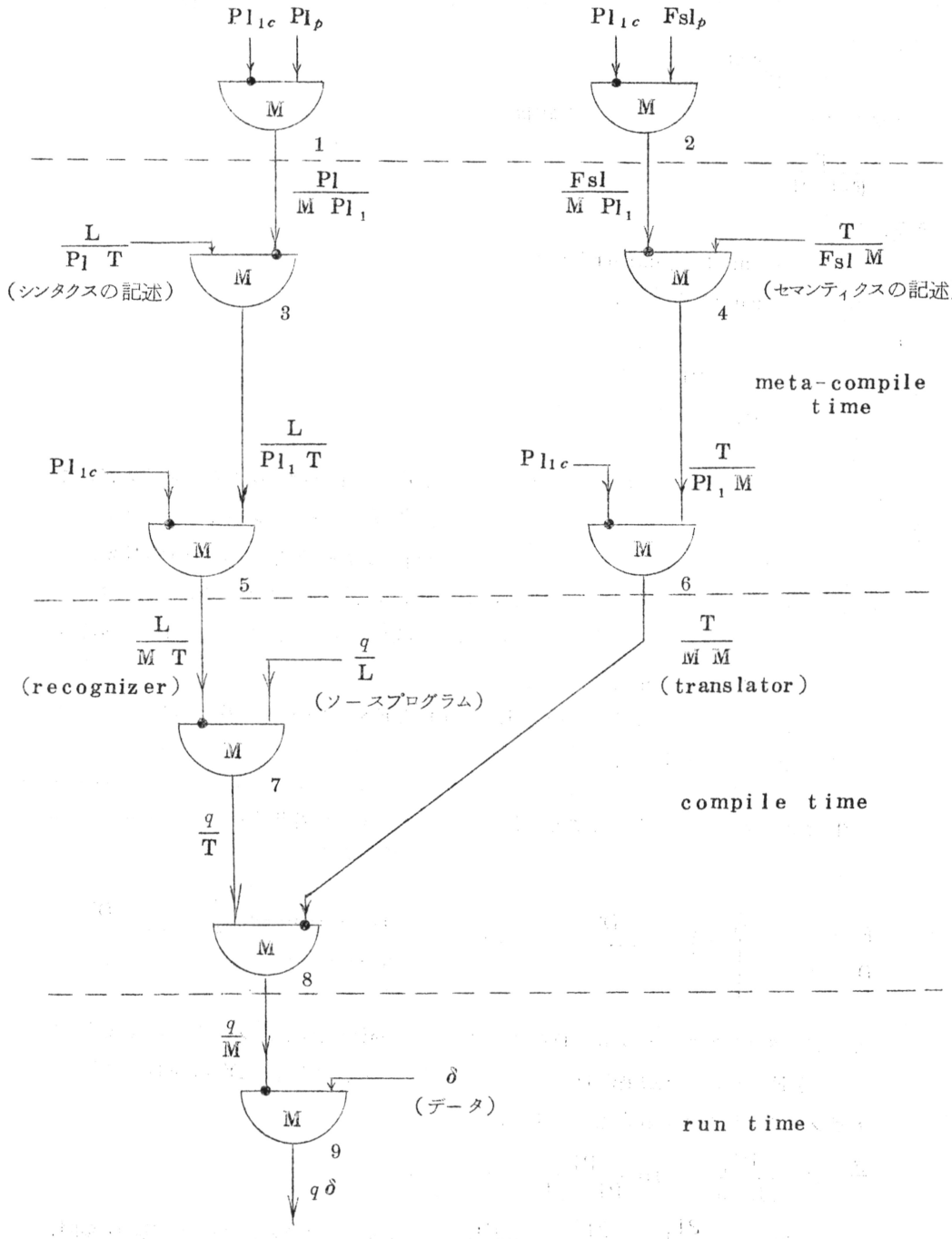

図2：プロセッサの相互関係

メタコンパイルタイム

図2のプロセス3と4は各々シンタクスとセマンティクスの記述をPL/1Wプログラムに翻訳する．プロセス5と6から各々作られるrecognizerとtranslatorは，実際は，図1におけるようにコンパイルタイムに互に影響し合う（即ち，一方が他方を呼出したり，木構造を格納する共通の記憶場所に作用し合ったりする）．従って，プロセス3と4の産物は同じ言語で書かれている方がそれ等のプロセスにとって便利である（PL/1Wとは無関係に作られた機械語プログラムをPL/1Wプログラムから呼出したり，その逆の呼出しをすることは非常にやりにくい）．

プロセス3（又は4）から直接機械コードを作らないで，PL/1Wプログラムを作り出した理由は次の2つである．

(1) PLプロセッサ及びFSLプロセッサの作成が楽である．

(2) プロセス3（又は4）の産物を言語設計者がオプティマイズしやすい．

しかし，そのために生成されるコンパイラ（図1の太線で囲まれたブロック）がPL/1Wで書かれることになり，次の3つの不都合が生じた．

(a) メタコンパイルタイムにPL/1Wコンパイラを走らせなければならない．5020 TSSにおいてはPL/1Wコンパイラはシリアル処理をしている．即ち，誰かがPL/1Wコンパイラを使っているとき他の人はそれが空くまで行列を作って待たなければならない．また，多数のユーザーが同時にTSSを利用している場合にはPL/1Wコンパイラが動き始めてからコンパイルを終了するまでに長時間（正確なデータはないが，500ステートメントで20分以上ということもある）待たなければならない．従って，PL/1Wコンパイラの使用回数が増えると仕事の能率が下がると考えられる．

(b) VITALのFSLでは生成されたコンパイラが使用するインデクスレジスタを宣言して使用できる．しかしPL/1には変数をインデクスレジスタに割付けるように指定する機能がないので，インデクス宣言を取除くようにFSLを変更しなければならなかった．

(c) コンパイルタイムに行なわれる主な計算の1つは番地計算である．そのためには半語演算ができれば十分である．HITAC 5020には能率の良い半語演算命令があるのにPL/1Wではそれを利用していない（PL/1Wで半語演算をするプログラムを書いてもその目的コードは2倍時間のかかる1語演算命令を使っている）．従って生成されるコンパイラの番地計算能率は良くない．

このフェイズが働くのは言語Lの文法をディバッグしてコンパイラを作るときだけである．従ってそれ以降のフェイズに比べると働く回数はかなり少ないと考えられるので(a)はあまり問題にならないと考えられる．しかし，(b)と(c)は速いコンパイラを作ろうとするときに障害となる．

コンパイルタイム

ソースプログラムをrecognizerが読込んで木構造を作り（プロセス7），それを

translatorが直接に機械コードに翻訳する（プロセス8）．図2ではこの2つのプロセスは別別に働くように書いてあるが，実際は交互に働く．このフェイズは直接機械コードを作るのでPL／1Wコンパイラを使う必要がないし，目的プログラムはPL／1W言語の制限（インデクスレジスタの指定ができない，インダイレクトビットをつけたり消したりできない，命令の合成ができない，等）を受けない．従って目的プログラムは，早くしかも言語設計者の思いのままに能率の良いものができる．

ランタイム

必要ならばデータを読込んだり書出したりしながら目的コードが走る．FSLは伝統的な番地方式を持つ計算機においてリエントラントでない目的プログラムを作り出すコンパイラを記述するための言語である．従ってここでのランコード（目的プログラム）もリエントラントではない．

2次元番地付け方式の計算機用又はリエントラントな目的プログラムを作るコンパイラ用にVITALを変更することは興味深い問題であると考えるが，我々はまだそれに手を付けていない．

3. FSLプロセッサにおける記憶場所割付け

FSLは，ソースプログラムに対応する木構造を機械コードに翻訳する過程を記述するための言語である．FSLプログラムにはソースプログラムに現われる変数に番地を割付ける操作及びその番地と属性を覚えておくためのテーブル，目的コードが生成されるべき記憶場所，目的コードを生成する操作，番地計算のための演算及び結果を記憶するための記憶場所，条件文の翻訳のために使われるスタック，等が含まれる．

FSLプログラムに含まれる記憶場所には3種類ある．それ等は，

c_1：コンパイルタイム（FSLプログラムが評価されるとき，即ちプロセス8）及びランタイム（FSLプログラムが評価されてできた目的プログラムが走るとき，即ちプロセス9）に参照（書込み又は読出し）されるもの．

c_2：コンパイルタイムだけに参照されるもの．

c_3：ランタイムだけに参照されるもの．

である．c_1とc_2はコンパイルタイムに，そしてc_1とc_3はランタイムにメモリーに確保されていなければならない．c_2とc_3は異なるフェイズで使われるのでメモリー上で重なってもかまわない．

c_1，c_2，及びc_3の番地は整数とみなされてコンパイルタイムに演算されることがあるので，整数型データと同じに扱える番地を持たなければならない（即ち，番地で参照できなければならない）．図2に示したとおり生成されたコンパイラはPL／1Wプログラムであるからc_1とc_2をPL／1Wの変数としてとると番地を整数型データとして扱えなくなる．[2] 特に5020 TSSは2次元番地付方式を採用しているので数値データと番地は全く異なっている．

c_1, c_2, 及び c_3に番地を持たせるために64k語（1k＝2^{10}）のメモリーを持つ計算機を想定し，そのメモリーに各々に属する記憶場所を割付けた．メモリーはコンパイルタイムとランタイムの2つのフェイズに存在する必要があるのでPL/1Wの外部アレイにした(図3参照)．

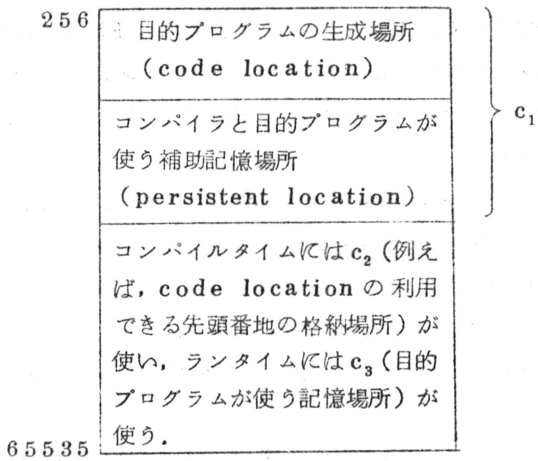

図3：記憶場所割付け

4. 入出力

ソースプログラムに含まれる入出力操作の処理に関するFeldman[4]の見解は次の文に示されている．「FSLは現在入出力操作を形式的に記述するための機能を備えていない．そのかわり，FSLシステムで記述される全てのソース言語が共通の入出力言語を含むようになっている．この入出力言語はPerlisのフォーマット言語[5]とよく似たものである．

このように決めた理由はいくつかある．入出力言語には，それが埋蔵されるソース言語よりもむしろ金物の方に強く依存するという性質があるように思われる．言語設計者がトランスレータを作成する際に入出力言語を完全に指定しなければならないようにしてしまうことは，我我のプロジェクトの目的に反することである．現在のシステムにおいて言語設計者が入出力に係りあうのは非常に表面的レベルにおいてのみである」．

ソース言語にフォーマット言語を埋蔵させるためにはPLプロセッサとFSLプロセッサの他にフォーマット言語用プロセッサを作成しなければならない．我々はそうしないで，FSLに入出力用ステートメント（数値又は記号データを1つ読め又はテレタイプに書け，等の原始的機能を持つもの）を追加するだけにした．それ等の機能を持つサブルーチンは5020 TSSライブラリに既に登録してあるので，ライブラリプログラムへのリンクを目的プログラムの中に生成するだけの作業で済んだ．そのため5020 TSS用VITALのユーザーは，入出力の処理に関して常に原始的レベルにまで係りあわなければならない．現在のVITALシステムを用いて

フォーマット言語用プロセッサをVITAL自身に追加することを考慮中である．

5. メタコマンド

5020 TSS用VITALのユーザーが端末からPLプロセッサ，FSLプロセッサ，及び生成されたコンパイラを呼び出すために用いるコマンドを各々PL，FSL，及びVITALと呼ぶ．この3つと5020 TSSのコマンドが文献1)においてメタコマンドと呼はれる範ちゅうに属する（VITALではFSLステートメントをコマンドと呼ぶので，それより一段レベルの高いものをメタコマンドとしている）．

ファイルの作成，更新，印字等は全て5020 TSS用コマンドで行なう．我々が作成しなければならなかったのは下記の3つだけである．

5.1 $PL(plf, sl, *)

引数：2個又は3個（，* は省略する場合もある）．

plf：PLプログラム（シンタクスの記述）の書いてあるファイル名．

sl：ソース言語名

ユーザー（言語設計者）がソース言語，例えばsl，のシンタクスをPLで記述し，それを例えばplfという名前のファイルとして登録しておいたとする．このとき，コマンド$PL(plf, sl, *)を実行するとPLプロセッサが呼出されてplfを処理し，slという名前の外部手続（PL/1Wプログラム）を作ってPLLISTというファイルに登録する．即ち，

$$sl = \frac{L}{Pl_1\ T}$$

である（図2プロセス3 参照）．ユーザーがslをオプティマイズしたければファイルPLLISTを印字し，必要な箇所を変更すればよい．

$PL(plf, sl)が実行された場合は更にPL/1Wコンパイラが働き，外部手続slを機械語におとしてslという名前のファイルとして登録する（図2プロセス5 参照）．

5.2 $FSL(fslf, sl, *)

引数：2個又は3個（，* は省略する場合もある）．

fslf：FSLプログラム（セマンティクスの記述）の書いてあるファイル名．

sl：ソース言語名

ユーザー（言語設計者）がソース言語，例えばsl，のセマンティクスをFSLで記述し，それを例えばfslfという名前のファイルとして登録しておいたとする．このとき，コマンド$FSL(fslf, sl, *)を実行するとFSLプロセッサが呼び出されてfslfを処理し，sl_1という名前の外部手続（PL/1Wプログラム）を作ってFSLLISTというファイルに登録する．即ち，

$$sl_1 = \frac{T}{Pl_1\ M}$$

である（図2プロセス4 参照）．以下も$PLの場合と同様である．

5.3 $VITAL(sl, sf, of)

引数：3個

sl ：ソース言語名

sf ：ソースプログラムの書いてあるファイル名．

of ：目的プログラムの書かれるべきファイル名．

ユーザー（ソース言語の利用者）がコンパイルしようとするソースプログラム（ソース言語slで書かれている）をsfという名前のファイルに登録しておき，それをコンパイルした結果をofというファイルに登録したいとする．このとき，$VITAL(sl, sf, of)を実行するとコンパイラslが呼出されてsfを処理し，目的プログラムを作ってofというファイルに登録する（図2プロセス 7，8 参照）．この目的プログラムを走らせたいときは5020 TSSコマンド$RUN(of)を実行すればよい（図2プロセス9）．

6. 作業の方法

VITALシステムの作成作業は図1から明らかなようにPLプロセッサ，FSLプロセッサ，③，及び④を作ることである（実際にはこの他にメタコマンドの作成も含まれる）．

PLプロセッサ作成作業を軽減するためにはrecognizerの言語によって決まる部分を小さくし，言語に依存しない部分を大きくすればよい．即ち，③をPL/1 Wで書いておき，PLプロセッサの産物①は③に対する呼出し系列になるようにすればよい．この場合PLプロセッサが簡単になる上にメタコンパイルが早くできる（①をPL/1Wプログラムではなくて PLプログラムに対応する表にし，それをコンパイルタイムにインタープリートするようにすれば，作業は更に簡単になる）．③を5020 TSSのライブラリに登録しておけば多数のユーザーが同時にrecognizerを使う（即ち，コンパイルする）場合に③はコアメモリに常駐する可能性がありドラム又はディスクからコアへのswapの回数を減らすことができる．呼出し系列にすることによって増大したリンクに要する時間はswapの回数を減らすことによってある程度吸収できると考えられる．従ってこの場合，コンパイルに要する時間が極端に増大するとは考えられない．FSLプロセッサについても同じことが言える．

しかし実際の5020 TSSシステムの事情は複雑であり，コンパイルタイムに関する上述の議論が通用するかどうか見当がつかない．また，作業者の好みや能力とも相まって我々のシステムは議論どおりにはできていない．

大きなソフトウェアを複数人で作る場合，仕事の分割と作業の分担が問題になる．我々の場合，作業を次の3つに分けた．

(イ) PLプロセッサの作成（図1③の作成も含む）．

(ロ) FSLプログラムに含まれるコードブラケットの内側の処理プログラムの作成（図1④の作成も含む）．

(ハ) FSLプログラム中のコードブラケットの外側の処理プログラムの作成.

(ロ)と(ハ)によりFSLプロセッサが完成する．FSLプログラムではコンパイルタイムにやることとランタイムにやることの区別がはっきりとなされており，その区別はコードブラケットと呼ばれる括弧で示される．ランタイムにやられることはコードブラケットの内側に書かれる．コードブラケットの外側でやる主な仕事は変数の識別，番地割付け，及び番地計算であり，内側でやる主なものは目的コードの生成である．従って作業(ハ)には機械コードの知識は不要である(もちろんFSLプロセッサの大まかなスペックができていることは仮定する)．作業(イ)もPL/1Wを用いてPL/1Wを目的プログラムとするプロセッサを作るのであるから機械コードの知識は不要である．

(イ)，(ロ)，及び(ハ)は観念的にもはっきりと別れており，ディバッグも各々独立にかなりの程度まで可能である．我々は3人で(イ)，(ロ)，(ハ)の各々1つづつを分担して作業を進めてきたが，その能率から判断するとこの分け方は妥当であったと考えられる．

FSLプロセッサの作成は文献1)に従がい下記の順序で行なった．

① PLによるFSLシンタクスの記述

$$\frac{\mathrm{Fsl}}{\mathrm{Pl}\quad\mathrm{T}} \qquad \cdots\cdots (1)$$

② FSLトランスレータの作成

$$\frac{\mathrm{T}}{\mathrm{Pl}_1\quad\mathrm{M}} \qquad \cdots\cdots (2)$$

③ PLプロセッサで(1)を処理する．

$$\mathrm{M}\frac{\mathrm{Pl}}{\mathrm{M}\quad\mathrm{Pl}_1}\,\frac{\mathrm{Fsl}}{\mathrm{Pl}\quad\mathrm{T}}=\frac{\mathrm{Fsl}}{\mathrm{Pl}_1\quad\mathrm{T}} \qquad \cdots\cdots (3)$$

(2)と(3)を結合したものがFSLプロセッサである．

7. 結言

VITALは，FeldmanのFormal Semantic Language[4]の流れをくむコンパイラコンパイラである．FeldmanとGriesのサーベイ[6]によれば，この種のコンパイラコンパイラはG−20とTX−2という計算機に対しては既に作成され，IBM360シリーズに対してはCarnegie，Stanford，及びLincoln Laboratoryの3カ所で作成されつつあるそうである．我国においても2，3カ所で作成されているという話を耳にしたことはあるが詳細は不明である．

本稿執筆の時点では，我々のシステムは，PL/1Wによるプログラミングがようやく完了し，簡単なテストはしたがディバッグは終了していないという状況である．従って，手書のコンパイラとの比較等の能率評価は行なっていない．PLプロセッサとFSLプロセッサの詳細，それ等の欠点，及び改良の余地等については別の機会に述べるつもりである．

本システムの作成にあたり，日立中研高橋延匡主任研究員と高徳一恵主任技師には5020 TSS使用上の便宜を計っていただいた．中田育男主任研究員及びその研究室の皆様にはPL／1Wについて御教示いただいた．特に，浜田穂積，霜田忠孝の両氏にはシステム構成及びPL／1Wによるプログラミングとディバッグ技術について貴重な御意見をいただいた．以上の方方に心から感謝の意を表します．

8. 参考文献

1) Mondshein, L.F., VITAL Compiler-Compiler System Reference Manual, Lincoln Lab MIT, 1967

2) 浜田穂積他，PL／1Wによるシステムの開発，情報処理学会第11回プログラミングシンポジウム，1970年1月

3) Sklansky, J., et al., A formalism for program transformation., J.ACM 15, 2 (April 1968), 165−175

4) Feldman, J.A., A formal semantics for computer oriented languages., Comput. Ctr., Carnegie Institute of Technology, 1964

5) Perlis, A J., A format language., Comm. of ACM, Vol.7, No.2, February, 1964

6) Feldman, J.A., Gries, D., Translator writing systems, Comm. of ACM, Vol.11, No.2, February, 1968