北海道大学工学部研究報告
第99号 (昭和55年)

Bulletin of the Faculty of Engineering,
Hokkaido University, No. 99 (1980)

Pascal コンパイラの HITAC VOS3 システムへの移植に伴う内部仕様設計と移植プロセス

魚住哲朗* 宮本衛市*

(昭和55年3月31日受理)

Internal Detailed Design and Transporting Process of Pascal Compiler to HITAC VOS3 System

Tetsuro UOZUMI Eiichi MIYAMOTO

(Received March 31, 1980)

Abstract

This paper describes the internal detailed design and the transporting process of the Pascal compiler which has been transported from FACOM 230-75 system to HITAC M180 system in Hokkaido University Computing Center. In the transporting process, at first, the so-called trunk compiler, in which machine-dependent parts are removed from the original compiler, is implemented, and then machine-dependent parts oriented to the target system are inserted into it. The source program of the new compiler is compiled twice in the old system. The produced object module is transported to the target system by a magnetic tape, and linked with the library routines to be an executable program.

1. まえがき

北海道大学大型計算機センターにおいて，昭和54年9月末に FACOM 230-75 計算機システムから HITAC M180 計算機システムへのシステム変更が行われた。これに伴い，従来 FACOM 230-75 システムの M-VI/VII オペレーティングシステム (以下旧システムと呼ぶ) の下で開発されていた Pascal コンパイラを，HITAC M180 システムの VOS3 オペレーティングシステム (以下新システムと呼ぶ) へ移植することが必要となった。本論文は，旧システムから新システムへの Pascal コンパイラの移植に関して，その内部仕様設計と移植プロセスについて述べたものである。

旧システム下での Pascal コンパイラは著者らの研究室で開発されたもので[1]，それ自身 Pascal で記述されている。一般に，コンパイラは特定の計算機に依存する部分と依存しない部分とに分けて考えることができ，後者が構文解析をする部分に対応するのに対して，前者は実行プログラムを働らかせる計算機に固有の機械語を生成する部分に対応する。旧システム下の Pascal コンパイラも基本的にはこのような構造を有しており，しかも Pascal で記述されているため，旧システム下の Pascal コンパイラから旧システムに関与した部分を削除して，特定の計算機に

* 情報工学専攻 情報システム工学講座

は依存しない，いわゆるトランクコンパイラを抽出し，それに機械語生成などの計算機依存部分を追加すれば新システムへの移植を行うことができる[2]。

移植に当って設けた目標は次の通りである。

(1) 旧システム下のコンパイラの持つ機能をすべて受続ぐ。

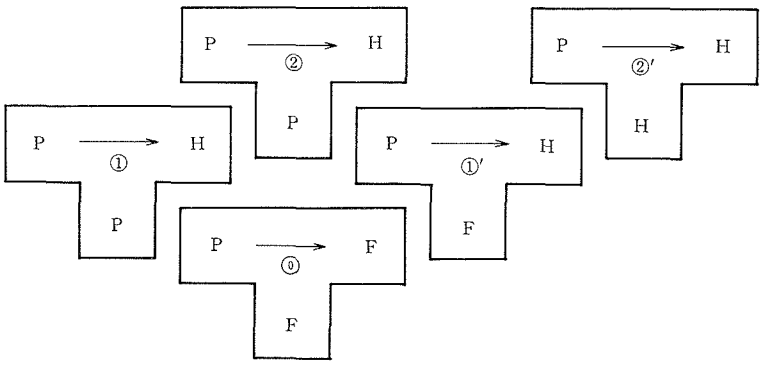
(2) ただし，移植時には，Pascal で記述された Pascal コンパイラは最小限，誤りなく翻訳することができること。

(3) 新システムの特長を考慮し，効率のよい機械語生成を行うこと。

後述するように，移植プロセスの大部分は旧システムを使って行われる。したがって，旧システムの稼動中に移植プロセスが完了しなければならず，もちろん (1) および (3) がこれまでの Pascal ユーザをサポートする上からも当然であるが，(2) の条件が満たされれば以後は新システム上で改良を進めてゆくことができる。

2. 移植のプロセス

移植のプロセスを T-ダイヤグラムで示すと図-1 のようになる。同図で 1 つの T 型の図は 1 つのコンパイラを表わしており，たとえば ① は Pascal で書かれたソースプログラムを HITAC システムの機械語に翻訳するコンパイラ (P→H) が Pascal で記述されていることを示している。それゆえ，⓪ が旧システム下で稼動しているコンパイラであり，②′ が目的とする新システム下でのコンパイラを意味する。図-1 は 2 段階の翻訳過程を示しており，第 1 段階では ① のコンパイラを Pascal プログラムとして旧システム下で稼動しているコンパイラ ⓪ で翻訳し，第 2 段階では前段階で得られた実行可能なコンパイラ ①′ で再度 ② で示すコンパイラを翻訳すると，目的とする新システム上で実行可能な Pascal コンパイラ ②′ が得られる。① および ② のコンパイラは旧システム用のコンパイラの計算機依存部分を新システム用に改訂したものである。両者は新旧両システムの文字コードが同一の EBCDIK コードのため基本的には同一のものであり，ただ翻訳後のオブジェクト・モジュールを ① では旧システム向けの，② では新システム向けのファイルを作成する点のみが異なるだけである。②′ は新システム向けのオブジェクト・モジュールとなっており，これを磁気テープを介して新システムへ移し，これに新システム側の入出力制御用などのライブラリと結合し，新システム下で実行可能なロード・モジュールに編成する。



P: Pascal
H: HITAC M 180 機械語
F: FACOM 230-75 機械語

図—1 移植プロセス

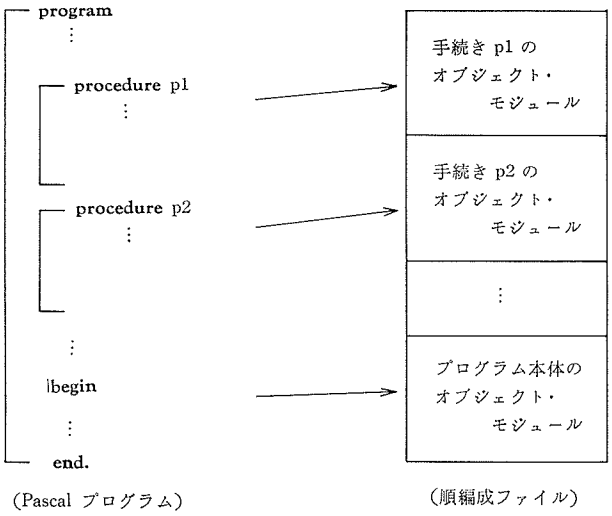
以上からもわかるように ②′ までの作成プロセスはすべて旧システムを用いて行われるので，新システム側での翻訳テストの結果誤りが発見され，その原因がコンパイラ自身にある場合には ① および ② を修正し，旧システム側で図-1 に示す翻訳プロセスが必要となる。ただし，この移植プロセスはほとんど Pascal という高級言語のレベルで行うので，デバッギングあるいはテストも比較的容易に行うことができる利点を有する。

3. オブジェクト・モジュールの生成

新システム用の Pascal コンパイラは，図-2 に示すように Pascal で書かれたソースプログラムを手続き，関数を単位として直接機械語に翻訳し，オブジェクト・モジュールとして，現われた順に順編成ファイル上に出力する。各手続きのオブジェクト・モジュールは HITAC システムの特徴を考慮し，命令語の列からなる部分と定数値を集めた部分から構成される。定数値としては，

(1) ソースプログラム上での定数値
(2) アドレス修飾に使用する定数値
(3) 他の手続きとの結合に用いる外部アドレス値

が含まれており，計算機から要求される語境界を考慮して配列する。



図—2　オブジェクト・モジュールの作成

4. レジスタの割当てと管理

コンパイラの計算機依存部分の作成における最も重要なことの一つが，生成する機械語におけるレジスタ類の取扱いである。新システムでは，16 個の汎用レジスタを持ち，レジスタ使用の自由度は大きいが，一方新システムのアドレス指定方式が旧システムよりは繁雑で，特に直接指定できるアドレスの変位値のために 12 ビットしか保有しておらず，レジスタによるアドレス修飾が絶えず必要となるなどの環境条件を考慮しなければならない。

新システムにおける Pascal コンパイラでは，上記の点を考慮して次に示すように各レジスタの役割を設定し，機械語の生成を行っている。

(1) *RG0*, *RG1*: 演算およびパラメータ設定のための作業用として。
(2) *RG2*: プログラム本体のデータセグメントの先頭アドレスを保持。
(3) *RG3*: 実行中の手続きのデータセグメントの先頭アドレスを保持。
(4) *RG4*: 実行中の手続きのプログラム・モジュールの先頭アドレスを保持。
(5) *RG5*: 実行中の手続きの定数領域の先頭アドレスを保持。
(6) *RG6*～*RG13*: 演算用，ベースレジスタ用，インデックス・レジスタ用およびレコードポインタ用として，後述するようにコンパイラが管理して割当てる。
(7) *RG14*: 作業用および分岐時の戻りアドレス設定のため。
(8) *RG15*: 作業用および分岐時の飛び先アドレス設定のため。

RG0 から *RG5* までと *RG14*, *RG15* の8個のレジスタはそれぞれ特定の目的のために割当てられているが，これは翻訳処理の複雑化を避け，さらに処理の効率化を図ったためである。一方，*RG6* から *RG13* までの8個のレジスタは多目的に使用されるため，それらの機能を考慮した効率的な管理が必要である。基本的にはレジスタの必要が生じたときに空いているレジスタを割当てるが，空きレジスタがないときには使用中の適当なレジスタを解除して割当てる管理方式を採用している。ただし，それぞれの使用目的に応じ，効率的な機械語の生成を考慮してその取扱い方には工夫をこらしている。以下，用途に応じたレジスタの割当て方式を要訳する。

(1) 演算用として 演算専用レジスタ *RG0* および *RG1* に対して補助的に用いられ，割当て要求に対して空きレジスタがある場合にのみレジスタが割当てられる。したがって，空きレジスタがない場合には，*RG0* または *RG1* が解除されて割当てられる。

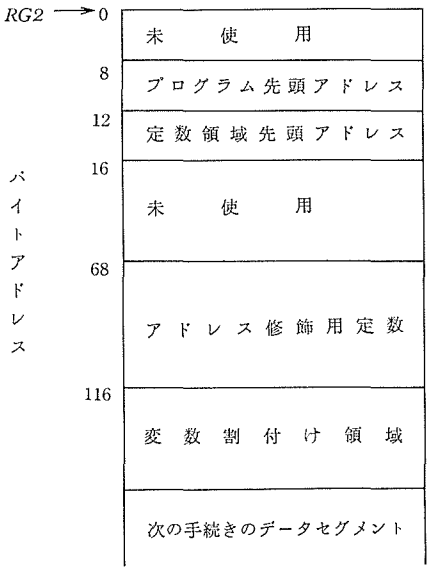
(2) ベースレジスタおよびインデックス・レジスタとして 割当て要求時に空きレジスタがなければ，すでに割当てられているレジスタの1つを解除して割当てる。ただし，解除するとき，演算用，インデックス・レジスタおよびベースレジスタの順で対象とする。

(3) レコードポインタとして これは **with** 文におけるレコード変数のアドレスを保持するためのもので，最大2個まで割当てられる。割当て要求時に空きレジスタがない場合には前項と同様な方式で解除するレジスタを決めるが，レコードポインタとして割当てられたレジスタは他の割当て要求によって解除されることがない。それはレコードポインタとしての性格上，途中で解除されてはその効果が十分発揮できないためである。一方，多くのレジスタがレコードポインタとして割当てられ，他の目的への割当てに柔軟性を欠くことを防ぐため，レコードポインタとしての割当ては2個に制限している。

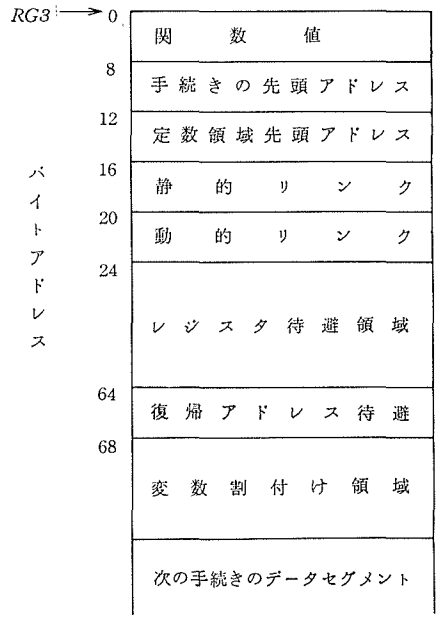
5. 実行時のデータ領域とその構造

プログラムの実行に先立ってデータ領域が割当てられ，プログラムで使用する変数等がこの領域に割付けられる。1つの手続きに対応するデータ領域をその手続きのデータセグメントと呼ぶと，手続きが呼ばれたとき対応するデータセグメントがスタック方式でデータ領域に割当てられ，手続きが終了するときに解放される。図-3および図-4はそれぞれプログラム本体および手続きのデータセグメントの構造を示したものである。同図で静的リンクは当該の手続きを呼んだ手続きのデータセグメントとリンクしてスタック構造を作っており，動的リンクは当該手続きよりレベルが1つ低いデータセグメントとリンクして，ブロック構造をなす大域的変数へのアクセスを可能にしている。

プログラム本体のデータセグメント内には，アドレス修飾用定数として 2^{12} の整数倍の定数が格納されている。これは新システムの機械語で直接指定できるアドレスの変位値が0から $2^{12}-1$



図—3　プログラム本体のデータセグメントの割付け



図—4　手続き・関数のデータセグメントの割付け

までであるので，基準番地から 2^{12} をこえる変位をもつアドレスを指定する必要があるときには，それに最も近い 2^{12} の整数倍をレジスタに設定し，インデックス・レジスタとしてアドレス修飾に加えてやる。

各手続きのデータセグメントには，レジスタの待避領域があり，当該手続きを呼んだ手続きが使用していたレジスタの内容を待避し，復帰時に回復させる。

6.　移植プロセスの経緯

移植プロセスは設計から完了まで約 3.5 人・月を要しているが，この程度の期間で移植を完了できたことは，移植プロセスがほとんど Pascal を用いて行うことができたことによる。図-5 は移植プロセスの経緯を線表で示したものである。各工程における作業内容をあげると次のようになる。

(1)　検討，設計

i)　新システムのハードウェアおよびオペレーティング・システムの検討および把握

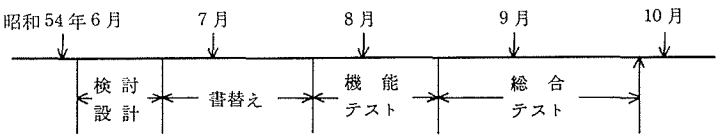
ii)　プログラムセグメントおよびデータセグメントの設計

iii)　レジスタの使用方式の検討

iv)　データ処理およびアドレス指定方式の検討

v)　コンパイラ内で必要な手続きの検討

(2)　書　替　え



図—5　移植プロセスの工程線表

i) 旧システム下の Pascal コンパイラからトランク・コンパイラの作成
ii) 計算機依存部分に関する手続きの設計およびコーディング

(3) 機能テスト
i) コーディングした新コンパイラの翻訳およびデバッギング
ii) 基本的な文に対するコンパイラの動作テストおよび生成する機械語の確認
iii) 比較的大きなテストプログラムに対する翻訳テスト

(4) 総合テスト
i) 入出力制御のためのライブラリの作成およびデバッギング
ii) 旧システムで作成した新システム向けのコンパイラのオブジェクト・モジュールを磁気テープを介して新システムに移植し，ライブラリと結合して動作テストを行う。
iii) Pascal コンパイラのソースプログラムに対して前項の動作テストを行う。すなわち，コンパイラ自身を翻訳できることを確かめ，さらにでき上ったコンパイラが正常に動作することを確認して移植を完了する。

7. あとがき

旧システムでの Pascal コンパイラとほぼ同等の機能をもつコンパイラの移植を短期間で遂行できたが，これは Pascal という高水準の言語レベルで移植プロセスを行うことができたためであり，その結果新システム向けのコンパイラもやはり Pascal で書かれているため，すぐれた保守性および拡張性をそのまま継承している。当然，他の計算機への Pascal コンパイラの移植に当コンパイラを利用することも同様に可能である。

今回移植したコンパイラが採用したレジスタの使用方法やアドレス指定方式は必ずしも最適なものとは考えることはできない。今後はより効率のよい実行プログラムを生成できるコンパイラに改良すると共に，データフロー解析[3]，誤り修復機能[4]，さらには Pascal 向けエディタ[5] などを包含した，いわゆるプログラミング・システム[6] に育てあげていく予定である。

文　献

1) 宮本，上原： 北大工学部研究報告，**85** (昭 52), p. 123–134.
2) Grosse-Lindemann, C. O. and Nagel, H. H.: Software-Practice and Experience, **6** (1976), p. 29–42.
3) 宮本： 情報処理学会論文誌，**21** (昭 55), 1, p. 8–14.
4) 宮本： 情報処理学会第 21 回全国大会講演論文集 (昭 55).
5) 宮本，浅見： 情報処理学会論文誌，**20** (昭 54), 6, p. 474–480.
6) 和田： 情報処理，**20** (昭 54), 8, p. 681–687.