PASCAL クロスコンパイラ・システムの実現

丹羽敏行・馬場文康*
情　報　工　学　科
(1980年9月16日　受理)

Implementation of the PASCAL Cross-compiler System

Toshiyuki NIWA and Fumiyasu BANBA*
Department of Information Engineering
(Received September 16, 1980)

PASCAL is a new programming language proposed and defined by N.Wirth. For its ample data structure and more sophisticated control structures, it now seems to have gained considerable popularity among many computing communities.

In our department, the PASCAL 8000 compiler has been adapted to the H-8450 comp uter. This version generates the H-8450 machine codes supported by the runtime system, and the execution time can be saved more than the P-code compiler. However, its compilation and execuion requires large memory, and its implementation on other computer is not easy.

We have designed the cross-compiler system for our minicomputers. This system is split into two independent sections, compiling PASCAL sources to the intermediate codes for the the virtual machine on the H-8450, and converting the intermediate codes to the machine codes on the target machine. The intermediate codes can be transfered from the H-8450 to the target machine under the computer network NITNET.

This intermediate language, named VAL, is also designed to be a common low-level language for our minicomputers.

1. まえがき

最近，PASCAL という新しいプログラミング言語で記述されたプログラムが目につくようになった。特に，マイコンの分野では BASIC に続くコンパイラ言語として注目を集めている[1]。

PASCAL は N. Wirth によって提唱され，その後1975年に Jensen と Wirth によって標準 PASCAL[2] として改訂された。その主な特徴は，系統的プログラミングの教育[3] と信頼性の高い効率のよい処理系の実現[4] に目標が置かれていることである。

本学情報工学科でも，PASCAL 8000[5][6] という処理系が H-8450 に移植[7]され，卒研や情報工学演習に使用されている。この PASCAL 8000 は標準 PASCAL の拡張版として設計され，その最初のコンパイラは東京大学大型計算機センタにおいて H-8800/8700 上で実現された。さらに，G. Cox らによって IBM 360/370 シリーズ用に改造され，広く使用されている。

この処理系は，H.H. Nägeli によって作成された Trunk PASCAL を移植したもので，機械語のオブジェクトを生成する完全なコンパイラである。したがって，この処理系を移植するためには標的マシン毎にコード生成部を書き換える必要がある。また，コンパイル時に必要な記憶容量が 135 KB 以上と大きく，ミニコンにはそのままでは移植はできない。

そこで，PASCAL で記述されたミニコン用のプログラムを H-8450 でコンパイルして，ミニコン用機械語オブジェクトを生成し，それをミニコン上で実行するクロスコンパイル方式が考えられる。ここで，コンピュー

*名古屋工業大学大学院工学研究科

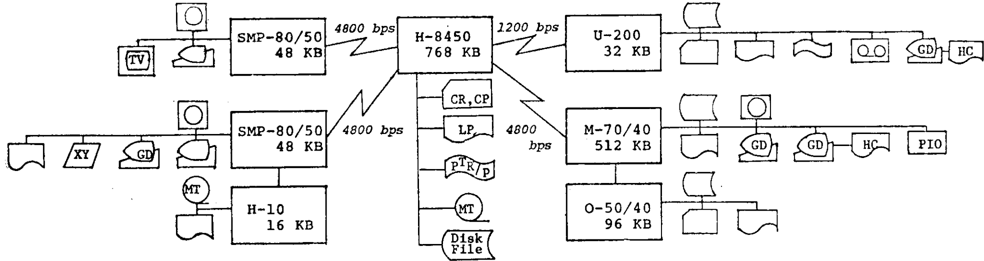

Fig. 1 Hardware configuration of NITNET

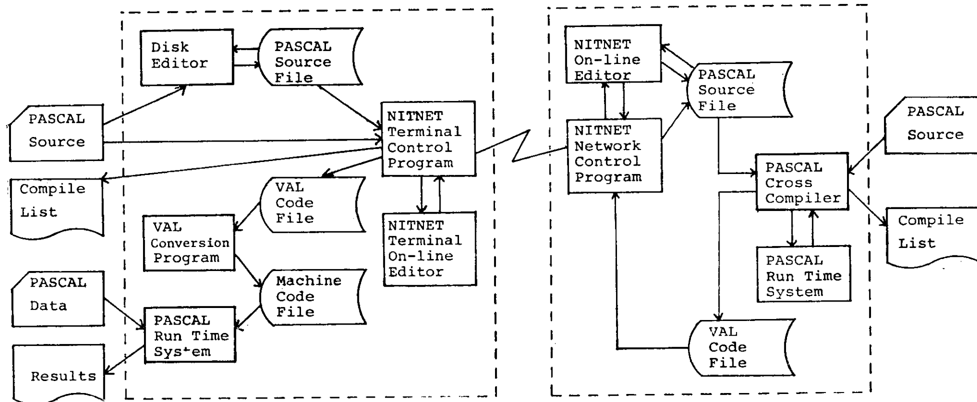

Fig. 2 Software structure and data flows of the PASCAL cross-compiler system

タ・ネットワーク NITNET[8] を利用することにより，ソースの投入とオブジェクトの転送及び実行をミニコン側で容易に行うことができ，またオンライン・エディタの使用によってデバッグの能率が格段に向上する。

さらに，多種のミニコンへの移植を容易にするため，クロスコンパイラのオブジェクトを共通の言語にすることができれば，移植コストは大巾に軽減されることになる。

以上のような観点より，NITNET を積極的に利用した PASCAL クロスコンパイラ・システムを設計したので，その概要を報告する。

2. システムの構成

2.1 ハードウェア構成

Fig. 1 に NITNET のハードウェア構成を示す。

各端末局と中央局 H-8450 は4線式の専用回線で接続されており，U-200 局は半2重 1200 bps の調歩同期式を，他の3局はいずれも半2重 4800 bps の SYN 同期式を採用している。

2.2 ソフトウェア構成

本システムのソフトウェア構成を Fig. 2 に示す。

太線の枠内が今回製作したプログラムである。NITNET のサポートする機能は，RJE, オンライン・エディタ，ファイル転送などがあり，PASCAL ソースや VAL コード及びクロスコンパイラの起動に利用する。

2.3 処理形態

本システムは NITNET のもとで動作するので，以下に示す多様な処理形態が可能である。

(1) PASCAL ソース・プログラムを任意の局から入力する。ソースは NITNET オンライン・エディタやディスク・エディタによって作成，修正が可能である。

(2) コンパイル・リストやエラー・メッセージを任意の局に出力する。

(3) クロスコンパイラの生成した VAL コードを任意の端末局で実行させる。VAL は各局共通であるから，プログラムの VAL コードがあれば，コンパイルせずにファイル転送によって，任意の端末局で実行できる。

最終的に得られるのは，端末局の機械コードで記述された PASCAL プログラムであるから，これを実行させる場合には NITNET は必要でない。

2.4 処理手順

PASCAL プログラムのコンパイルと実行の概要を，

```
//JOB
//RJIN
!JOB
!#PASCALV

    PASCAL Source

/*
!END
//PARAM    RF=DDC1/¥PASOBJV
//PARAM    OUT=DD00/VALOBJ,ENQ=YES
//RJOUT
//FASP
/INPUT DD00
/FILE VALOBJ
//PRBL
//EXEC

    PASCAL Data

//END
```

Fig. 3 Typical card deck from the target machine

端末局から PASCAL ソースを入力し端末局で実行する場合を例にして，Fig. 2を用いて説明する。この時のジョブコンを Fig. 3に示す。

(1) PASCAL ソースを '//RJIN' コマンドによって，NITNET を通して，中央局へ転送する。

(2) 中央局でバッチジョブとしてクロス・コンパイルを実行し，VAL コード・オブジェクトを生成する。

(3) '//RJOUT' コマンドによって，VAL コードを再び NITNET を通して端末局へ転送する。

(4) VAL コードを端末局の機械コードに変換する。

(5) ランタイム・システムの支援によってプログラムを実行する。

Table 1. Routines for code generation

Routine name	Parameter	Function
GENOP	(1) Operation (2) Grobal attribute (3) Local attribute	Generate code to operate local attribute on grobal attribute
GENCSP	(1) Entry name of run time system (2) Parameter	Generate code to call system program
GENJMP	(1) Grobal attribute (2) Condition (3) Label	Generate code to conditional jump

Table 2. Entry names of Run Time System

SIN	Standard arithmetic function	RET	Return
EXP	Exponential	VARPROC	Procedure call
WB	Write byte	OPENINPUT	Open input file
WC	Write character	GETCH	Get character
WI	Write integer	WRITLN	Write line
WS	Write strings	CLOCK	Get clock
WR	Write real	TIME	Get time
RC	Read character	GET	Get next file element
RI	Read integer	OPEXT	Open external file
RR	Read real	CLEXT	Close external file
RL	Read logical	OPLOC	Open local file
RS	Read string	CLLOC	Close local file
		PAGE	New page
		HALT	Halt
		MESSAGE	Message print out
		LONGJUMP	Long jump

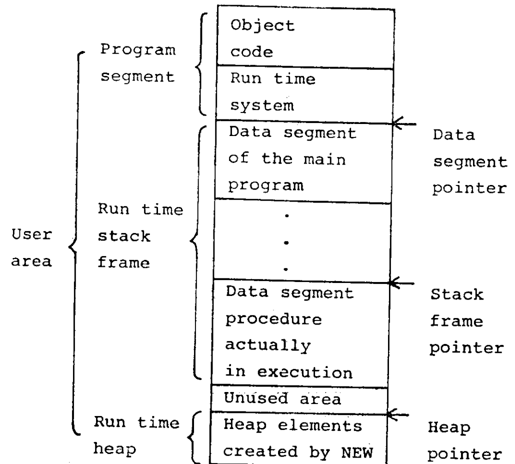

Fig. 4 Memory map for the PASCAL cross-compiler on H-8450

3. クロスコンパイラの構成

3.1 概　要

クロスコンパイラ自身は PASCAL 8000 で記述されており，アセンブリで書かれたランタイム・システムでサポートされている。このクロスコンパイラは，今後の情報工学科としての研究を考慮して，高い拡張性，保守容易性，移植性を重視した構成をとっている。

PASCAL の言語仕様は1パス方式に適しているため，このクロスコンパイラも1パス方式である。つまり，コンパイルは PASCAL ソースを読み込みながら，並行して構文解析，各種テーブルの作成，コード生成が行なわれる。まず字句解析部で，PASCAL ソースを少しずつ読み込み，そのトークンを一つずつ返していく。宣言部解析では，渡されたトークンを構文図に従って解析しな

から，各種テーブルの生成を行ない，本体解析では，トークンの解析と各種テーブルの参照を行ないながら必要なコード生成を行なっていく。したがって，構文解析はトップダウン方式を採用している。

3.2 コード生成

このクロスコンパイラは VAL で記述されたコードを生成するため，コード生成の主な機能は，式の評価に関するコードと分岐命令に関するコードの生成である。前者は，コード生成に必要な諸情報を attribute と呼ぶレコードで定義し，演算の種類とその対象となるattribute を用いて実現している。後者は，単にラベルを用いているため，アドレス計算や飛び先番地を後で埋め込むことなどはしていない。

コード生成に関する主な手続きを Table 1 に示す。GENCSP で用いるランタイム・システムのエントリ名は Table 2 に示す。

3.3 実行時のメモリ構成

PASCAL プログラムの実行時におけるメモリ・マップを Fig. 4 に示す。

スタック領域には，現在実行中の手続きのためのデータ・セグメントが積まれていく。データ・セグメントには，手続き内の引数，ダイナミック・リンクのための領域がとられる。

PASCAL では，手続きの再帰呼び出しが許されるので，一般に，データ・セグメントのスタティック・リンクとダイナミック・リンクを採用している[9]。スタティック・リンクとは，そのデータ・セグメントに対応する手続きのレベル上の親へのポインタであるが，このコンパイラでは，特定のレジスタをこれに割当てるため必要ない。ダイナミック・リンクとはデータ・セグメントのスタック上で，一つ前のデータ・セグメントへのポインタである。

4. 仮想アセンブリ言語 (VAL) と変換プログラム

4.1 中間言語

コンパイラのオブジェクトとして中間言語を用いる場合の第1の利点は，他機種への移植や新言語処理系の実現が容易となることである。

中間言語としては，入出力端が自由な UNCOL，入力

Table 3. Hardware data for virtual and real machines

			VAL	U-200	O-50	M-70
Registers	General registers		16-bit x 16	16-bit x 8	16-bit x 8	16-bit x 8
	Floating arithmetic registers		64-bit x 2	—	64-bit x 2	64-bit x 2
	Stack pointers		General registers	0	0	0
Data length	Integer		1, 2, 4 bytes	1, 2, 4 bytes	1, 2, 4 bytes	2, 4 bytes
	Real		4, 8 bytes	—	4, 8 bytes	4, 8 bytes
	Boolean		1, 2, 4 bytes	1, 2 bytes	2 bytes	2 bytes
	String		1-256 bytes	—	1-256 bytes	1-256 bytes
Instructions	Length		—	2, 4, 6 bytes	2, 4, 6 bytes	2, 4, 6, 8 bytes
	Memory access		byte	byte	word (2 bytes)	word (2 bytes)
	Addressing	Register direct	○	○	○	○
		Register indirect	○	○	—	—
		Memory direct	○	○	○	○
		Memory indexed	○	○	○	○
		Immediate	○	○	○	○
		Memory indirect	—	—	○	○
		Memory indexed-indirect	—	—	○	○
	No. of instructions (No. of mnemonics)	Integer operations	10	25	22 (76)	38
		Real operations	10	—	22 (58)	23
		Boolean operations	17	25	20 (71)	43
		Stack operations	2	—	—	2
		Others	11 (26)	13 (39)	36 (86)	139 (151)
		Total	50 (65)	63 (89)	100 (291)	245 (257)
Assembly	No. of operand formats		7	9	14	16

Table 4. Assembly statement for load operation

Data length	Addressing mode	VAL	U-200	O-50	M-70
1	Register direct	@L Rd, Rs, 1	MVB Rs, Rd	LBR Rd, Rs	—
	Register indirect	Rd, (Rs), 1	(Rs), Rd	LB Rd, 0(Rs)	LDB Rd, 0, Ry
	Memory direct	Rd, Src, 1	Src, Rd	LB Rd, Src	LDB Rd, Src
	Memory indexed	Rd, Src(Rs), 1	Src(Rs), Rd	LB Rd, Src(Rs)	LDB Rd, Src, Ry
	Immediate	Rd, =I, 1	=I, Rd	LBI I	{LDI Rd_1, I_1 / LDIE Rd, I_1}
2	Register direct	@L Rd, Rs, 2	MV Rs, Rd	LR Rd, Rs	MOVE Rd, Rs
	Register indirect	Rd, (Rs), 2	(Rs), Rd	L Rd, 0(Rs)	LDL Rd, 0, Ry
	Memory direct	Rd, Src, 2	Src, Rd	L Rd, Src	LDL Rd, Src
	Memory indexed	Rd, Src(Rs), 2	Src (Rs), Rd	L Rd, Src (Rs)	LDL Rd, Src, Ry
	Immediate	Rd, =I, 2	=I, Rd	LI I	LDIL Rd, I
4	Register direct	@L Rd, Rs, 4	—	LDR Rd, Rs	—
	Register indirect	Rd, (Rs), 4	—	LD Rd, 0(Rs)	LDD Rd_2, 0, Ry
	Memory direct	Rd, Src, 4	LD Src, Rd	LD Rd, Src	LDD Rd_2, Src
	Memory indexed	Rd, Src(Rs), 4	Src(Rs), Rd	LD Rd, Src(Rs)	LDD Rd_2, Src, Ry
	Immediate	Rd, =I, 4	—	LDI I	—
Remarks			Data transfer from memory to memory is also permitted for MVB, MV and MVI.	Memory indirect is permitted for LBK, LK and LDK.	Memory indirect is also permitted for LDB, LDL and LDD. Rd_1: R_0, R_1, R_2, R_3 Rd_2: R_0, R_4 R_X: R_2, R_3 R_Y: R_2, R_3, R_7 $0 \leq I_1 \leq 7$

として多様な言語を想定し出力は特定のマシンである MU_5 の CTL[10]，逆に特定言語を多様な機種に移植することを目的とした OCODE や P-code など[11]があるが，いずれも標的マシン・コードへの翻訳プログラムの実現が大変である。

そこで，出力端をアーキテクチャのよく似たミニコンに限定し，移植性の他にこれらのミニコン共通の低位言語とすることを目的に，仮想アセンブリ言語とその変換プログラムを設計した。

4.2 仮想マシンの構成

筆者らの身近にあって，本システムの標的マシンと考えられるミコニンのアーキテクチャを Table 3 に示す。3 機種の重要な相違点として，次のことが挙げられる。

(1) U-200 には，実数演算と10進演算機能がなく，メモリ間接指定が許されていない。

(2) M-70 の汎用レジスタはレジスタ番号により機能が異なる。例えば，インデックス・レジスタとして使用できるのは R_2 と R_3 及び R_7 のみであり，4-byte 整数演算は R_0, R_1 及び R_4, R_5 の上でしか実行できない。

したがって，仮想マシンとしては U-200 と M-70 の間の共通機能を基本にした。ただし，実数演算やストリング処理については必須と考え，仮想マシンに含めることとし，M-70 ではサブルーチンで処理することにした。その結果を表中左端の欄に示す。

ここで重要な拡張は次の 5 点である。

(1) 16-bit 汎用レジスタを多くし，それぞれがスタック・ポインタとしても使用できるようにした。これで，いままでレジスタ数の少なさを呪いながら，メモリへの退避と復元を繰返していたのが見掛け上なくなった。

(2) アドレス指定でメモリ間接をなくし，スタック・ポインタ兼用の汎用レジスタの間接指定で対処することにした。

(3) 命令数をできるだけ少なくした。一般には，演算データの長さに応じて，特に M-70 では対象レジスタ番号によっても，命令コードやニーモニック・コードが異なり，プログラムの作成と読解を困難にする第 1 原因となっている。

(4) オペランド形式も整理し，覚え易くした。

(5) スタック操作命令を加え，しかもレジスタだけでなく，メモリをも操作対象にできるので，プロブラムが簡潔になる。

4.3 VAL の仕様

4.2 で述べた仮想マシン用の仮想アセンブリ言語を

```
PASCAL   K:=I*10+J

VAL      @L      R0,I,4     ……①
         @M      R0,=10,4   ……②
         @A      R0,J,4     ……③
         @ST     R0,K,4     ……④

U-200    MV      I+2,R1    ⎫
         MV      I,R0      ⎬ ①
         BNES    *+4       ⎪
         CI      =0,R1     ⎭
         MVI     =0,R00    ⎫
         MV      =10,R10   ⎬ ②
         CALL    MD        ⎪
         DC      A(R00)    ⎭
         AL      J+2,R1    ⎫
         BLS     *+4       ⎪
         AI      =1,R0     ⎬ ③
         A       J,R0      ⎪
         BNES    *+4       ⎪
         CI      =0,R1     ⎭
         MV      R1,K+2    ⎫ ④
         MV      R0,K      ⎭

O-50     LD      R0,I
         MDI     R0,10
         LDR     R0,R2
         AD      R0,J
         STD     R0,K

M-70     LDD     R0,I
         LDIL    R5,100
         LDIL    R4,0
         STD     R4,R00
         MPD     R0,R00
         ADD     R0,J
         STD     R0,K
```

Fig. 5 Assembly statements generated from the PASCAL cross-compiler

VAL (Virtual Assembly Language) と名附け，その記述例を Table 4 に示す。この形をとる命令は 32 種と多く，この言語によるプログラムの記述と理解を容易にしている。

4.4 標的アセンブリ言語への変換

標的マシンのアセンブリ言語への変換は，標的マシンの標準マクロ機能を利用して実行される。PASCAL クロス・コンパイラのオブジェクトを変換した例を Fig. 5 に示す。ここで，U-200 には 4-byte 整数の乗除算命令は存在しないので，サブルーチンで処理されることになる。

名機種の特性差による変換時の問題点として，

(1) ストア命令でのコンディション・コードの立ち方
(2) ビット比較でのコンディション・コードの立ち方
(3) 4-byte 整数演算でのレジスタ指定の制限
(4) シフト命令でのシフト数の指定方法

などが挙げられるが，PASCAL のオブジェクトでは無関係となるよう考慮されている。

5. あとがき

現在クロスコンパイラ及び変換プログラムを，ミニコン側は U-200 の上で実現中であるが，すでに次のような問題点が見つかっている。

(1) ミニコンのマクロ機能が不足のため，変換プログラム（マクロ定義）が冗長となり，

(2) 展開速度も非常に低く，

(3) また，PASCAL オブジェクト中に頻繁に出現する 4-byte 整数演算におけるコンディション・コードの設定が困難で冗長になり易い。

したがって，既成のマクロ機能に頼らない高速な変換プログラム，及び条件コードの保存に関する補助引数の導入が必要と考えられる。

また，他の標的マシンへの移植に伴う速度と大きさの非効率化率や移植コスト率などによるシステムの評価，及び分割コンパイル機能や各システム対応機能の追加拡張によるセルフ・コンパイラの実現は今後の課題である。

最後に，日頃熱心に御討論いただく名古屋大学工学部情報工学科の福村教授と吉田助教授，ならびにファコムハイタック（株）の楠葉氏に感謝致します。

文　献

1) 安田寿明：Tiny PASCAL 移植のすすめ，bit, 1980-7, pp. 92-132
2) K. Jensen, N. Wirth: PASCAL User Manual and Report, Lecture Notes in Computer Science, Springer-Verlag, 1974
3) N. Wirth: Algorithms + Data Structures = Programs, Prentice-Hall, 1976
4) 疋田輝雄：コンパイラのキットを用いた PASCAL の移植，日経エレクトロニクス，1976, 12. 13, pp. 100-131
5) 米田，疋田：PASCAL プログラミング，サイエンス社，1979
6) G. Cox, J. Tobias: PASCAL 8000 IBM 360/370

Version Reference Manual, 1978

7) 藤社彰：NITNET 応用プログラムの開発研究，名古屋工業大学大学院工学研究科修士論文，1980

8) 丹羽，井口，藤社：科内ネットワーク・システム NITNET の開発と応用，名古屋工業大学学報，第31巻，1979，pp. 329–336

9) U. Ammann: On Code Generation in a PASCAL Compiler, Software-Practice and Experience, Vol. 7, 1977, pp. 391–423

10) D. Morris, R.N. Ibett: The MU_5 Computer System, Macmillan Press, 1979

11) E.F. Elsworth: Compilation via an intermediate language, The Computer Journal, Vol. 22, No. 3, 1979