UDC681, 3. 06：800, 92

PASCAL言語とそのコンパイラ

Programming Language PASCAL and its Compiler

渡辺　勝*・安藤友久*

Masaru WATANABE and Tomohisa ANDO

新しい計算機の汎用言語PASCALが注目を浴びている，PASCAL言語の特徴とプログラム例に簡単にふれたあと，PASCALコンパイラの作成について述べ，本所のFACOM 230－55機にこのコンパイラを移した実験例について詳細に述べたものである．

1. PASCAL言語

プログラム言語PASCALはN. Wirth（スイス，チューリッヒの連邦工業大学）によって1970年に発表された新しい計算機用言語である[1]，それにつづく数年間の使用経験にもとづいて，若干の改良が加えられ[2]，今日に到っている．その間この言語のもつすぐれた性質が，世界各地の大学で計算機学者の注目するところとなり，多くの計算機に対してこの言語が実現され，使用されている．このように比較的短期間に普及がなされたことは，これまでひろく使用されてきたFORTRAN, ALGOLあるいはPL/Iなどにあきたりないだけでなく，それらの持つ欠陥が鋭く指摘されていることの証拠である．

PASCALは計算機言語の歴史に新しい一ページを開いたものであり，プログラム言語の理想へ導く一つの道標であるといえよう．

PASCALの設計にあたっての基本的な考え方はつぎのようなものである．

(1)　ALGOLを基礎におき，最近の構造化プログラミング（Structured Programming）の理論に適応した構文を採用したこと．

(2)　豊富なデータ構造を加えることにより，言語の適用範囲を拡大したこと．

(3)　言語構成が簡潔かつ合理的であり，"正しい"プログラムの作成および保守が容易であること．

2. PASCALの特徴

PASCAL言語については，すでに分かり易い入門書が出ているのでそれを参考にして頂くことにして詳細は省略する[3][4][10]．以下にPASCALの特徴を簡単にふれおこう．

まずデータの型として，従来の整数，実数などのスカラーおよびそれらの配列に加えて，新たにセット，レコード，ポインタおよびサブレンジなどの新しい型（type）が導入されている．さらにプログラマが望ましい型を定義できる機能があって，たとえば，

```
type color = ( red, yellow, green, blue )
```

によって，colorという新しい型（スカラー）を定義できる．

そこで

```
var c: color
```

とすれば，cはcolorの型を持つ変数と宣言され

```
c: = red
```

によって，cはredの値をもつことになる．

colorの型をもつ変数は，上に定義した四つの"原色"のいずれか一つしか値としてとれないが，その組合せの色たとえば橙色（orange）をあらわしたい時に，セット型の変数が使われる．すなわち，

```
var hue1, hue2 : set of color;
hue 1 : = [ red, yellow ];
hue 2 : = [ red, blue];
```

などとすれば，橙色や紫色があらわせる．

つぎにレコード型の変数につき述べよう．データ処理に適した構成を持ち，PL/Iなどでいう構造変数（structure）に対応している．すなわちレコードはいくつかの要素から構成されていて，各要素は異なった型であってもよい．たとえば，

```
type person = record name: alfa;
                     age: integer;
                     married: boolean
              end
```

ここでalfaは先頭が英字である英数字のストリングをあらわす．レコード型変数の各要素に値をセットするには

```
var X, Y: person;
X. name: = 'Watanabe';
X. age: = 56;
```

などとすればよい．

PASCALのプログラム構成につき述べよう．それはALGOLのプログラム構成によく似ているが，いくつかの改良が加えられている．

実行文の種類には次のようなものがある．

*東京大学生産技術研究所　第3部

代入文
if - then - else 文
case 文
for 文
while 文
repeat 文
with 文

などである．if 文の拡張である case文，構造化を容易にして go to 文の使用を避けるために用意された while 文，repeat 文などが採用されている．さらに go to 文も用意されているが，例外処理やエラー処理などの特別な場合以外は使用しないように勧告されている．

プログラムがブロック構造を持つこと，それに関連して変数の有効範囲（scope rule）が定められていることなどは ALGOL と同様である．また，サブルーチンは procedure あるいは function として定義されることも ALGOL と同じであるが，パラメタの扱いで従来問題のあった点を整理して，call by value と call by reference の2種のパラメタに限定し，call by name は廃止している．

3. プログラム例

PASCALを用いてプログラムを作成した例題として，“8－Queen”の問題を図1に示してある．8×8のチェスの盤の各行のどの列に Queen をおいてゆけばよいかを試行錯誤して求めてゆくプログラムであり，Nと Hは行と列の添字を，また Xは Queenをおかれた位置，COL, UP, DOWN は対応する列，右上り対角線，右下り対角線上にすでに Queenがおかれているか否かをあらわす配列である．

プログラムの初めにある GENERATE という手順は，一つの行のどの列におくかを定めるサブルーチンであり，これを反復して recursive に呼出し，すべての行について求まった時にその配置を印刷するようにしている．プログラムを読みやすくするために行の先頭をずらせたり，“……”によってコメントを入れてある点，あるいは repeat 文の使い方などに注意されたい．また（．　．）記号は［　］の代りに使用したものである．

4. PASCALのコンパイラ

PASCAL 言語のコンパイラを各種の計算機に作ろうとすると，対象となる計算機に固有な命令やアドレス方法に合わせて，コード生成を行うことが必要である．このような不便をさけて，容易に移し変えの可能な，いわゆる“portable”コンパイラを作るために採用されている有力な一つの方法はつぎのようなものである．

```
"EIGHT QUEEN PROBLEM"
VAR N,K:INTEGER;
    X:ARRAY(.0..7.) OF INTEGER;
    COL:ARRAY(.0..7.) OF BOOLEAN;
    UP:ARRAY(.-7..7.) OF BOOLEAN;
    DOWN:ARRAY(.0..14.) OF BOOLEAN;
PROCEDURE GENERATE;
  VAR H:INTEGER;
  BEGIN
    H:=0;
    REPEAT
      IF "SQUARE(N,H) FREE" COL(.H.)&UP(.N-H.)&DOWN(.N+H.) THEN
      BEGIN "SET QUEEN ON SQURAE(N,H)"
        X(.N.):=H;
        COL(.H.):=FALSE; UP(.N-H.):=FALSE; DOWN(.N+H.):=FALSE;
        N:=N+1;
        IF "BOARD FULL" N=8 THEN
          BEGIN "PRINT CONFIGURATION"
            K:=0;
            REPEAT WRITE(X(.K.):1); K:=K+1 UNTIL K=8;
            WRITE(EOL)
        END
        ELSE GENERATE;
        N:=N-1; "REMOVE QUFEN FROM SQURE(N,H)"
        DOWN(.N+H.):=TRUE;  UP(.N-H.):=TRUE;  COL(.H.):=TRUE
      END;
      H:=H+1
    UNTIL H=8
  END;
  "INITIALIZE EMPTY BOARD"
  BEGIN
    N:=0;
    K:=0;
    REPEAT COL(.K.):=TRUE; K:=K+1 UNTIL K=8;
    K:=0;
    REPEAT UP(.K-7.):=TRUE;  DOWN(.K.):=TRUE;  K:=K+1 UNTIL K=15;
    GENERATE
  END.
```

図1　PASCAL のプログラム例

PASCALで書かれた原プログラムをいきなり対象とする計算機の命令語に翻訳しないで，いったん中間の言語に変換するコンパイラを作る．つぎにこの中間言語に対するインタプリタを各計算機ごとに作成すれば，それぞれの計算機で実行が可能となる．

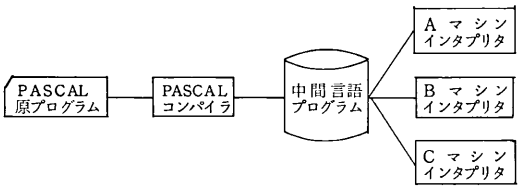

図2　ポータブルPASCALコンパイラ概念図

さらにPASCALコンパイラもPASCAL自身で書いおいて，これを中間言語に変換したものを作って各ユーザに提供するものとする．ユーザ側では，上記のインタプリタをコンパイル段階とオブジェクトの実行段階の2度にわたって使用するようにすれば，各計算機ごとのPASCALコンパイラが実現したことになる．

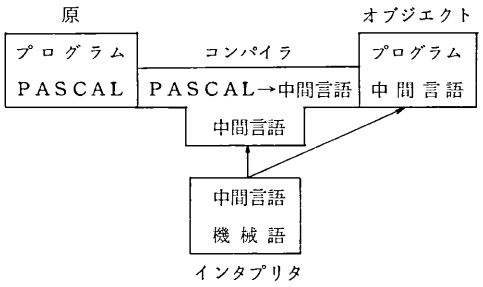

図3　インタプリタによるPASCALプログラムのコンパイルと実行

このような考え方で，スイス連邦工業大学のAmmann等によって，ポータブルなPASCALコンパイラが作られた[7][8]．このコンパイラは中間言語として，仮想的なスタック式計算機を用いているが，これについては次節で述べる．ここではコンパイルの手法について簡単にふれておこう．

コンパイルの手法は基本的に"反復下降法"（recursive descent）といわれている方法を用いている．PASCAL言語は比較的簡単な文法構成であり，BNF（Backus Normal Form）記法を用いて記述できる．その根(root)から始めて順に文法要素をたどってゆき端記号（terminal）に進むtop-downの解析法を行い，コンパイルが進行する．すなわちプログラムはALGOL同様のブロックから構成され，ブロックは一連のステートメントから成るわけで，それに対応してブロックを解析する手順，おのおのステートメントを解析する手順が，順次に反復して呼出される．例としてステートメントの解析手順をif文について説明しよう．if文の文法はBNF記法に従えば，

＜if statement＞::=
　if ＜expression＞ then＜statement＞
　if ＜expression＞ then＜statement＞
　else＜statement＞

であり，これに対応してif文を解析する手順は，つぎのようなPASCALプログラムで書ける．

```
procedure if statement;
begin
  expression;
  if current symbol = then symbol
    then get next symbol else error;
  statement;
  if current symbol = else symbol then
      begin
            get next symbol;
            statement
      end
end "if statement";
```

この手順の中で，expressionという手順が呼ばれており，またstatement手順が反復（recursive）に呼ばれている．また手順の呼出しに際しては1字先読みが行われている．このような方法でワンパスのコンパイラが作成されている．生成されたコードの出力はあらかじめ用意されたバッファに出力され，ブロックの処理が終了したところでオブジェクトファイルに書き出すようになっている．

コンパイラはすべてPASCAL自身で書かれていて，カードにして約3,500枚から成っている．

5. スタック計算機

コンパイラの中間言語はスタック方式をとる仮想計算機の命令コードとして定義されている．以下この仮想計算機（Stack Computer略してSC）について説明しよう．

SCは四つのレジスタと主記憶から成る．各レジスタはつぎの通りである．

PC　プログラムカウンタ
SP　スタックポインタ
MP　マークポインタ
NP　ニューポインタ

SP, MP, NPは主記憶の構成に関係して定義される．主記憶は命令を格納する部分CODEと，データを記憶する部分STOREとに分けられる．PCはCODEに対する指標（インデクス）であり，SPはSTOREに対する指標であるが，詳しくは後述する．

CODE部に格納される命令の1語は三つの部分から成る．命令コードをあらわすOP部，アドレスおよびその修飾を指定するP部とQ部がある．各部分の長さ（ビッ

ト数）は，図4に示す構成になっている．

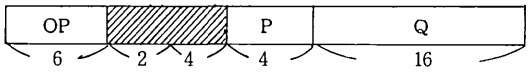

図4　スタック計算機の命令コード

命令の種類とその意味は附録にまとめておく[9]．

STORE部に格納されるデータの種類には整数，実数論理数，文字などがあり，それぞれビット長が異なっている．これらのデータを能率よく記憶につめこむためには，語長を可変にして，各語の属性をタグ（識別部）によって指定するようにしなければならない．そこで記憶の使い方としては少々無駄になっても取扱いを容易にするため，同一のデータ長として最大のものに合わせてしまうようにした．PASCALコンパイラで使われている最大語長の変数は，言語で使用するSYMBOLの組をあらわすセット変数であり最大51ビットから成る．そこでデータ1語の大きさは64ビットに統一している．

つぎにプログラムの実行にあたってSTORE部が動的にどのように使用されているかを説明しよう．STORE部は図5に示すようにスタックとヒープに分けられ，スタックはアドレス0から増加する方向に，ヒープはアドレス最大から減少する方向に使用される．現在使用中のスタックおよびヒープの先頭を指示するポインタがそれぞれSP，NPである．

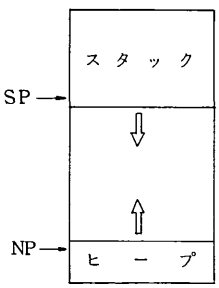

図5　STOREの使い方

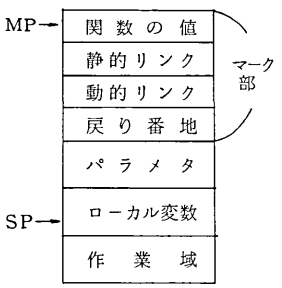

図6　データセグメント

プログラムの進行にともなって，手順（procedure）または関数が呼ばれると，スタックのあたまにその手順または関数のための新たなデータセグメントが作り出される．データセグメントは図6に示す構成になっていて，つぎの各領域から成る．

マーク部　図に示す4語

パラメタ　手順または関数のパラメタ

ローカル変数　手順または関数の内部だけで使う変数

作業域　手順または関数の処理にともなって使用する作業域

このデータセグメントにアクセスするために設けられているのがマークポインタMPである．マーク部に設定される静的リンクおよび動的リンクの意味はつぎの通りである．プログラムのテキスト内での手順の配置の順序に従って各セグメントをリンクしているのが静的リンクである．プログラムの実行にともなって手順が呼出される順序に従って各セグメントをリンクしているのが動的リンクである．たとえば図7のような構造を持つプログラムがあったとしよう[6]．このプログラムの進行にともなって，主プログラムM→手順Q→手順R→手順Pの順に呼出される．各手順が呼出された時のデータセグメントの配置は図8に示される．図において右側の矢印が静的リンクを，左側の矢印が動的リンクをあらわしている．静的リンクは上位にある手順内のデータにアクセスする時に使用される．動的リンクは手順が終了して上位の手順に復帰する時に使用され，戻り番地は呼出し側の手順のマーク部に設定されている．また呼出された方が関数の場合，その返す値は呼出された側のマーク部の先頭におかれる．（図6参照）

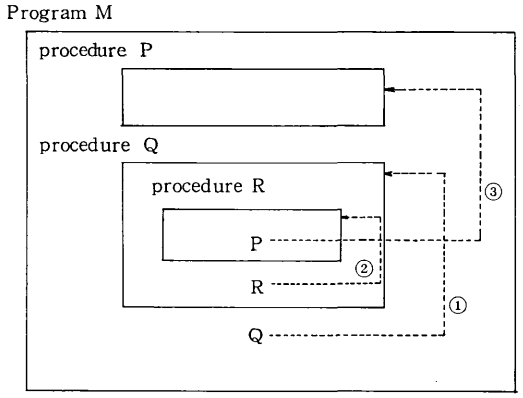

図7　入れ子構造の手順をもつプログラム

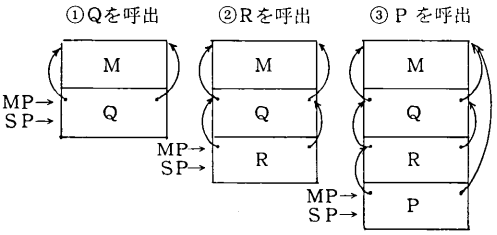

図8　手順呼出時のスタック内データセグメント配置

スタック計算機では通常の計算機にみられる演算レジスタを持たず，演算は原則としてスタックの最上部のセルに対して行われる．したがって演算命令はアドレスを持たない．（附録参照）アドレスを持つのはデータのロードおよびストアに関係した命令の場合である．

6. インタプリタ

インタプリタは上記のスタック計算機の各命令を解釈し実行するプログラムであり，インタプリタの作成が本報告の主な作業であった．

インタプリタの主ルーチンXEQTERの一部を図9に示してある．命令の演算を指定するOPおよびアドレスに相当するP，Q部を分解抽出するINSTRUCTION

DECODEの部分，および指定された演算を行うためそれぞれのルーチンに分岐するINSTRUCTION EXECUTIONの部分などに注意されたい．命令の実行に先立ってPCプログラムカウンタに1を加えている．

インタプリタは当初FACOM230－55機のアセンブラで組むことを考えたが，記述の容易さや明確さ，他機

```
      SUBROUTINE XEQTER(*)
      COMMON /AREA/ CODE
      REAL*8 CODE(10000),STORE(6300)
      INTEGER*4 ISTORE(12601),ICODE(19999)
      EQUIVALENCE (CODE(7387),STORE(1)),(STORE(1),ISTORE(2))
      EQUIVALENCE (CODE(2), ICODE(2))
      COMMON /POINTR/ SP,NP,MP
      INTEGER*4 SP,NP,MP
      COMMON /ROCOM/ INPUT*,GETEOF,PUTEOF
      INTEGER*2 INPUT*
      LOGICAL GETEOF,PUTEOF
      INTEGER*4 PC,KODE,OPP,OP,P,Q,I,J,K,AD,IAD
      INTEGER*2 WORK(12),WBUF(24),FMT(4)
      INTEGER*4 IDINT,BASE
      REAL*8 RDREAL
      INTEGER*2 GRD,CHR
      I=0
      ISTORE(1)=0
      DO 10000 I=1,12601
10000 ISTORE(I)=0
      GETEOF=.FALSE.
      PUTEOF=.TRUE.
      CALL READO
      IF (.NOT. GETEOF) ISTORE(8)=ORD(INPUT*)
      PC=0
      SP=-1
      MP=0
      NP=6300
C.
C.        INSTRUCTION DECODE
C.
  100 KODE=ICODE(PC)
      IF (KODE .LT. 0) KODE=KODE+2147483647+1
      OPP=KODE/65536
      Q=KODE-OPP*65536
      IF (Q .GE. 32768) Q=Q-65536
      OP=OPP/256
      P=OPP-OP*256
      OP=OP/4
      IF (ICODE(PC) .LT. 0) OP=OP+32
C.
C.        INSTRUCTION EXECUTION
C.
      PC=PC+1
      GOTO ( 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16,17,18,19,20,
     2       21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,
     4       41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,
     6       61,62,63),OP
      IF (OP .NE. 0) GOTO 63
      Q=BASE(P)+Q
      GOTO 1
    1 CALL PUSH(&99)
      STORE(SP)=STORE(Q)
      GOTO 100
    2 Q=BASE(P)+Q
      GOTO 3
    3 STORE(Q)=STORE(SP)
      SP=SP-1
      GOTO 100
    4 Q=BASE(P)+Q
      GOTO 5
    5 CALL PUSH(&99)
      ISTORE(SP*2)=Q
      GOTO 100
    6 Q=ISTORE(SP*2-2)
      STORE(Q)=STORE(SP)
      SP=SP-2
      GOTO 100
    7 CALL PUSH(&99)
      ISTORE(SP*2)=Q
      IF ((P .NE. 1) .AND. (P .NE. 3)) ISTORE(SP*2)=-1
```

図9　インタプリタ主ルーチン

種との互換性なども考慮して，大部分はFORTAN で作成した．アセンブラの持つキメの細かい処理をFORTRANで記述できるよう，つぎの諸点には特に工夫をこらしてある．

(1) 一語の命令の中の各フィールドの抽出処理

命令コードの分解処理を行うINSTRUCTION DECODEに見られるように，FORTRANではシフト命令が使えないので，整数の乗除算命令で代行している．

(2) 入出力命令の実行

PASCALの入出力文をコンパイルするとスタック計

```
0137    WRS→  506 AD=ISTORE((SP-2)*2)
0138              K=ISTORE((SP-1)*2)
0139              J=ISTORE(SP*2)
0140              IF (K .LE. J) GOTO 594
0141              K=K-J
0142              DO 595 I=1,K
0143          595 CALL WRITE(' ')
0144              GOTO 593
0145          594 J=K
0146          593 DO 592 I=1,J
0147          592 CALL WRITE(CHR(ISTORE((AD+I-1)*2)))
0148              SP=SP-3
0149              GOTO 100
0150    WRI→  508 J=ISTORE(SP*2)
0151              K=ISTORE((SP-1)*2)
0152              ENCODE(12,1000,WORK) K
0153         1000 FORMAT(I12)
0154              DECODE(12,1001,WORK) WBUF
0155         1001 FORMAT(12A1)
0156              IF (J .LE. 0) J=12
0157              IF(J .LE. 12) GOTO 590
0158              J=J-12
0159              DO 591 I=1,J
0160          591 CALL WRITE(' ')
0161              J=12
0162          590 J=13-J
0163              DO 589 I=J,12
0164          589 CALL WRITE(WBUF(I))
0165              SP=SP-2
0166              GOTO 100
0167    WRR→  509 J=ISTORE(SP*2)
0168              IF (J .LT. 8) GOTO 585
0169              IF (J .LE. 24) GOTO 587
0170              J=J-24
0171              DO 588 I=1,J
0172          588 CALL WRITE(' ')
0173              J=24
0174          587 K=J-7
0175              ENCODE(8,2000,FMT) J,K
0176         2000 FORMAT('(E',I2,'.',I2,')')
0177              ENCODE(24,FMT,WORK) STORE(SP-1)
0178              DECODE(24,2001,WORK)WBUF
0179         2001 FORMAT(24A1)
0180              DO 586 I=1,J
0181          586 CALL WRITE(WBUF(I))
0182              SP=SP-2
0183              GOTO 100
0184          585 CALL MESSAG(15)
0185              GOTO 99
0186    WRC→  510 J=ISTORE(SP*2)-1
0187              IF (J .LE. 0) GOTO 583
0188              DO 584 I=1,J
0189          584 CALL WRITE(' ')
0190          583 CALL WRITE(CHR(ISTORE((SP-1)*2)))
0191              SP=SP-2
0192              GOTO 100
0193    RDI→  511 AD=ISTORE(SP*2)
0194              ISTORE(AD*2)=RDINT(1)
0195              GOTO 580
0196    RDR→  512 AD=ISTORE(SP*2)
0197              STORE(AD)=RDREAL(1)
0198              GOTO 580
0199    RDC→  513 AD=ISTORE(SP*2)
0200              ISTORE(AD*2)=ISTORE(8)
0201              CALL READ0
0202              GOTO 580
```

図10　インタプリタにおける入出力命令の実行
WRA, WRI, WRR, WRC；ストリング，整数，実数，文字の各出力
RDI, RDR, RDC；整数，実数，文字の各入力

算機の入出力命令に変換される．その実行は基本的にはFORTRANの入出力文によって行うが，つぎの手順をとっている．

入力に関しては，文字単位の入力ルーチンREAD，数字および符号の読み取りルーチンRDDIG, RDSIGNおよびこれらをもとにした整数，実数の入力ルーチン

```
0001            LOGICAL FUNCTION RDSIGN(I)                                      PSC0
0002            COMMON /ROCOM/ INPUT*,GETEOF,PUTEOF                             PSC0
0003            INTEGER*2 INPUT*                                                PSC0
0004            LOGICAL GETEOF,PUTEOF                                           PSC0
0005            INTEGER*2 BLANK/'  '/, EOL/Z1540/, PLUS/'+ '/, MINUS/'- '/      PSC0
0006            RDSIGN=.TRUE.                                                   PSC0
0007         40 IF (GETEOF) RETURN                                              PSC0
0008            IF ((INPUT* .EQ. BLANK) .OR. (INPUT* .EQ. EOL)) GOTO 30         PSC0
0009            IF (INPUT* .EQ. MINUS) GOTO 20                                  PSC0
0010            IF (INPUT* .NE. PLUS) RETURN                                    PSC0
0011         10 CALL READO                                                      PSC0
0012            RETURN                                                          PSC0
0013         20 RDSIGN=.FALSE.                                                  PSC0
0014            GOTO 10                                                         PSC0
0015         30 CALL READO                                                      PSC0
0016            GOTO 40                                                         PSC0
0017            END                                                             PSC0
```

```
0001            SUBROUTINE RDDIG                                                PSC0
0002            COMMON /ROCOM/ INPUT*,GETEOF,PUTEOF                             PSC0
0003            INTEGER*2 INPUT*                                                PSC0
0004            LOGICAL GETEOF,PUTEOF                                           PSC0
0005            COMMON /RDCOM/ LZ,LI,TZ,NZP                                     PSC0
0006            INTEGER*2 LZ,LI,TZ                                              PSC0
0007            INTEGER*4 NZP                                                   PSC0
0008            LOGICAL COLLEC                                                  PSC0
0009            INTEGER*2 ZERO/'0 '/,NINE/'9 '/                                 PSC0
0010            NZP=0                                                           PSC0
0011            LZ=0                                                            PSC0
0012            LI=0                                                            PSC0
0013         10 IF (GETEOF .OR. (INPUT* .NE. ZERO)) GOTO 20                     PSC0
0014            LZ=LZ+1                                                         PSC0
0015            CALL READO                                                      PSC0
0016            GOTO 10                                                         PSC0
0017         20 TZ=0                                                            PSC0
0018            COLLEC=.TRUE.                                                   PSC0
0019         30 IF (GETEOF .OR. (INPUT* .LT. ZERO) .OR. (INPUT* .GT. NINE)) RETURNPSC0
0020            IF (INPUT* .NE. ZERO) GOTO 50                                   PSC0
0021            IF (COLLEC) COLLEC=.FALSE.                                      PSC0
0022            TZ=TZ+1                                                         PSC0
0023         40 CALL READO                                                      PSC0
0024            GOTO 30                                                         PSC0
0025         50 DECODE(1,100,INPUT*) INPUTD                                     PSC0
0026        100 FORMAT(I1)                                                      PSC0
0027            IF (.NOT. COLLEC) GOTO 60                                       PSC0
0028            NZP=10*NZP+INPUTD                                               PSC0
0029            LI=LI+1                                                         PSC0
0030            GOTO 40                                                         PSC0
0031         60 NZP=NZP*10**(TZ+1)+INPUTD                                       PSC0
0032            LI=LI+TZ+1                                                      PSC0
0033            TZ=0                                                            PSC0
0034            COLLEC=.TRUE.                                                   PSC0
0035            GOTO 40                                                         PSC0
0036            END                                                             PSC0
```

```
0001            INTEGER FUNCTION RDINT*4(I)                                     PSC0
0002            COMMON /RDCOM/ LZ,LI,TZ,NZP                                     PSC0
0003            INTEGER*2 LZ,LI,TZ                                              PSC0
0004            INTEGER*4 NZP                                                   PSC0
0005            LOGICAL IPLUS,RDSIGN                                            PSC0
0006            IPLUS=RDSIGN(1)                                                 PSC0
0007            CALL RDDIG                                                      PSC0
0008            IF (NZP .EQ. 0) GOTO 10                                         PSC0
0009            RDINT=NZP*10**TZ                                                PSC0
0010            IF (.NOT. IPLUS) RDINT=-RDINT                                   PSC0
0011            RETURN                                                          PSC0
0012         10 RDINT=0                                                         PSC0
0013            RETURN                                                          PSC0
0014            END
```

図11　入力命令のインタプリトに使用されるサブルーチン各種

```
0001            INTEGER FUNCTION CHR*2(I)                                       PSC(
0002            INTEGER*4 I                                                     PSC(
0003            INTEGER*2 CHRTBL(66)                                            PSC(
0004            DATA CHRTBL/Z1540,'A','B','C','D','E','F','G','H','I',         PSC(
               1          'J','K','L','M','N','O','P','Q','R','S',             PSC(
               2          'T','U','V','W','X','Y','Z','0','1','2',             PSC(
               3          '3','4','5','6','7','8','9','+','-','*',             PSC(
               4          '/','(',')','$','=',' ',',','.','#','_',             PSC(
               5          '%',':','>','"','|','&','@',Z3740,'<','>',            PSC(
               6          '?',',',';',Z0C40,Z0D40/                               PSC(
0005            IF ((I .LT. 0) .OR. (I .GT. 65)) GOTO 10                        PSC(
0006            CHR=CHRTBL(I+1)                                                 PSC(
0007            RETURN                                                          PSC(
0008          10 CALL MESSAG(17)                                                PSC(
0009            STOP 4                                                          PSC(
0010            END                                                             PSC(

0001            INTEGER FUNCTION ORD*2 (INPUT)                                  PSC0
0002            INTEGER*2 INPUT,ORDTBL(256)                                     PSC0
          C.                      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F PSC0
0003            DATA ORDTBL/ 21*61,                                             PSC0
               1                  0,                                           PSC0
               2                 33*61,                                        PSC0
               3                       57,  8*61,                              PSC0
               4             45,  9*61,                 61,47,58,41,37,54,     PSC0
               5             55,  10*61,                   43,39,42,63,62,     PSC0
               6             38,40,  9*61,                 46,50,49,59,60,     PSC0
               7                    10*61,                51,52,56,48,44,53,  PSC0
               8                    16*61,                                     PSC0
               9                    16*61,                                     PSC0
               A                    16*61,                                     PSC0
               B                    17*61,                                     PSC0
               C                  1, 2, 3, 4, 5, 6, 7, 8, 9,   7*61,          PSC0
               D                10,11,12,13,14,15,16,17,18,    8*61,          PSC0
               E                   19,20,21,22,23,24,25,26,    6*61,          PSC0
               F             27,28,29,30,31,32,33,34,35,36,    6*61           PSC0
0004            ENCODE(2,100,I) ORDTBL(22),INPUT                                PSC0
0005          100 FORMAT(2A1)                                                   PSC0
0006            ORD=ORDTBL(I+1)                                                 PSC0
0007            RETURN                                                          PSC0
0008            END                                                             PSC0
```

図12　PASCAL 内部コード→ EBCDIC 変換（CHR）
EBCDIC→PASCAL 内部コード（ORD）

```
0001            SUBROUTINE READO                                                PSC0
0002            COMMON /ROCOM/ INPUT*,GETEOF,PUTEOF                             PSC0
0003            INTEGER*2 INPUT*                                                PSC0
0004            LOGICAL GETEOF,PUTEOF                                           PSC0
0005            INTEGER*2 EOT/Z3740/                                            PSC0
0006            CALL READ(INPUT*,&10)                                           PSC0
0007            RETURN                                                          PSC0
0008          10 IF (GETEOF) GOTO 20                                            PSC0
0009            GETEOF=.TRUE.                                                   PSC0
0010            INPUT*=EOT                                                      PSC0
0011            RETURN                                                          PSC0
0012          20 CALL MESSAG(1)                                                 PSC0
0013            STOP 6                                                          PSC0
0014            END                                                             PSC0
```

図13　入力命令基本ルーチン（READ 0）

RDINT, RDREAL などを作成した.

出力に関しては，文字単位の出力ルーチン WRITE を基本にして,整数や実数の内部データは,まず ENCODE 文により文字（数字）に変換し，さらに DECODE 文により 1 字ずつ取り出して出力するようにしている．特に実数の出力に対しては，プログラムで指定した桁の長さに応じて，動的に変更可能なフォーマット FMT を使用して行っていることに注意されたい.

(3)　EBCDIC⇄PASCAL 内部コードの変換

FORTRAN プログラムで入出力される文字は，計算機内部では EBCDIC コードで表現される．これに対し PASCAL コンパイラにおける内部コードは表 1 に示す独自のコードが使われている．（PASCAL コンパイラを最初に作った計算機 CDC 6600 の内部コードに準拠したためか?）

出力に際してはすべて文字として出力されるので，PASCAL 内部コード→ EBCDIC 変換を行う．（CHR サブルーチン）入力の場合は，整数や実数は直接内部 2 進表示に変換され，文字入力の場合だけ EBCDIC→ PASCAL 内部コードの変換が必要となる．（ORDサブ

表1　PASCAL内部コード（8進）

上＼下	0	1	2	3	4	5	6	7
0	eol	A	B	C	D	E	F	G
1	H	I	J	K	L	M	N	O
2	P	Q	R	S	T	U	V	W
3	X	Y	Z	0	1	2	3	4
4	5	6	7	8	9	+	−	*
5	/	(	)	$	=		,	.
6	'	_	%	:	#	'	|	&
7	@	eom	<	>	?	¢	¬	;
10	ff	cr						

eol : end of line　　ff : form feed
eom : end of message　　cr : cariage return

ルーチン）

(4)　セット演算

PASCALの変数の型にセット（set）があることは既に述べた．セット型の変数に対しては，union(論理和)，intersection（論理積），difference（論理差），inn（セットのメンバテスト）などの演算が行われるが，これらはセット変数をビットのパターンとして，各ビットごとに指定された演算を行えばよい．しかしこの種の演算はFORTRANではきわめて困難である．したがってセット演算に限って，それぞれに対応したFASP（アセンブラ）サブルーチンを用意した．

7. インタプリタのための主記憶割付

インタプリタをFORTRANプログラムとして作成し実行する場合の主記憶（実計算機の）割付を考えてみよう．我々の使用したFACOM 230−55機におけるFORTRAN−Sの場合，主記憶はプログラムセグメントとデータセグメントに分けられ，おのおのは最大128 Kバイトまで使用可能である．

実際に作成したインタプリタのプログラム部分の大きさは46,160バイトになったが，これは十分余裕がある．一方インタプリタの使用するデータ部分は大別して次の三つの部分から成る．

(1)　実行の対象となるスタック計算機のプログラムCODE部（これはインタプリトすべきコード群であり，インタプリタに対してはデータとなる）

(2)　スタック計算機が使用するスタックおよびヒープから成るSTORE部

(3)　インタプリタプログラムが直接使用する各種変数（CODEとSTORE以外），ファイルの入出力のためのバッファ，その他システム用のデータ領域

データ部を構成する各部分の大きさは図14に示すようになった．これらはつぎのようにして決められたものである．まずCODE部については，PASCALコンパイラのコードが全部収容できる大きさが必要である．変数・バッファ等の領域はインタプリタ作成によって決まってくる．その残りが使用可能なSTORE部となるわけであり，図14の場合は1語8バイトとして6,301語とってある．

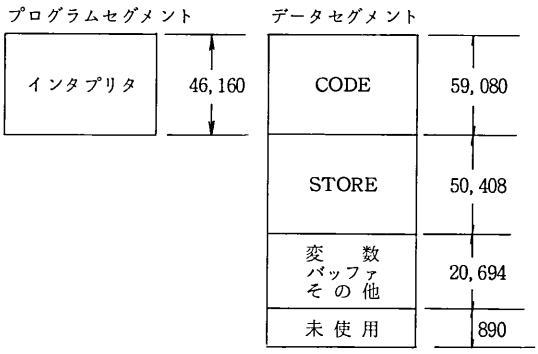

図14　インタプリタ実行時の主記憶割付

ふつうのPASCALプログラムのコンパイルおよび実行の際は，これで足りると考えられるが，PASCALで書かれたPASCALコンパイラ自身をコンパイルするには不足である．この場合の大きさ（STOREのみ）は116,208バイトになったことが報告されている[9]．

以上のような記憶の割付によってインタプリタを実行する際に問題となった点を二，三注意しておこう．

第1はスタック計算機のプログラムが使用する定数群の位置である．これらの定数群はCODEの末尾の部分に入れるように構成されている．そこでこの定数へのアクセスは負の添字を持ったSTOREの配列要素として行わなければならない．そのためCODEの最後とSTOREの先頭とをEQUIVALENCE文で結合しておいて（図9），STORE（負の添字）の形でCODE部にある定数データを読み出すようにしている．このような使い方は文法書では保証していないが，実際試みて幸いにもうまく使えることがたしかめられた．

第2はコンパイラのソースファイルや中間言語のオブジェクトファイルなど複数のファイルの入出力用バッファの取り方である．上述したようにデータ領域が手狭なので，これらのバッファは個々に設けないで，共通バッファ一つだけとして共用することにより領域の節約をはかり，データ領域を制限内におさめることができた．

8. ローダとアセンブラ

以上のインタプリタによってPASCALプログラムをコンパイルしたり，実行したりする場合，PASCALのコンパイラまたは中間言語のオブジェクトを主記憶（のCODE部）にロードしなければならない．PASCALコンパイラの方はコアイメージの形にしてあるので，比較的簡単なローダ（LOADER）によってロードできる．しかしユーザのPASCALプログラムをコンパイルして出力されるオブジェクトプログラムの方は，使用者が見

やすいように，スタック計算機命令のいわば“アセンブラ”コードの形をとっている．すなわち命令コードやアドレスは記号や10進数で示されているので，これを変換してコアイメージの形に直すことが必要である．このようなスタック計算機用のアセンブラがなければならない（ASMBLR）．また定数（16ビット以上の整数，実数，ストリング，セット）をロードする命令の場合，それらの定数を命令にくっつけて出力しているので，これらの定数を分離して，前節で述べたCODE部の末尾に配置し“間接”ロード命令（LDI）を用いてロードするよう変換する．これもASMBLRの仕事の一部である．

このようにしてユーザのPASCALプログラムをコンパイルし，実行する過程はつぎのフローによって行われる．

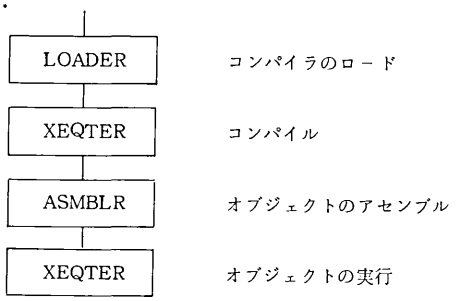

図15　PASCALプログラムの実行過程

9. プログラム構成と実施結果

インタプリタ（XEQTER），ローダ（LOADER），アセンブラ（ASMBLR）およびそれらに従属するサブルーチン群の関係を示したものが図16である．XEQTER

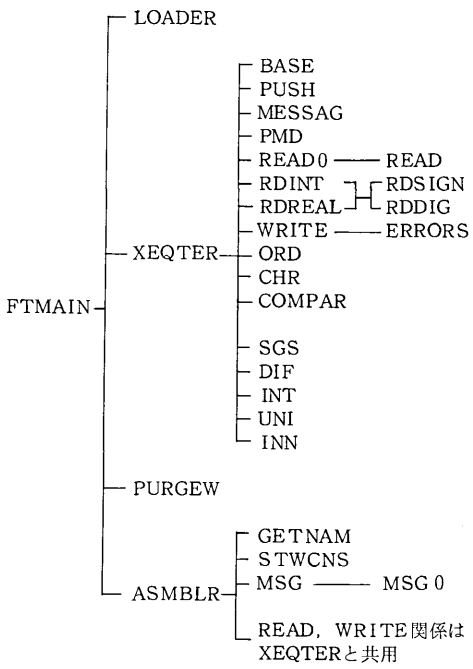

図16　PASCALコンパイラ構成プログラム

とその下にあるサブルーチンをふくめたFORTRANカードは1,586枚，LOADERとASMBLRが各々28枚，300枚である．このほかFASPで作成したサブルーチンが105枚，全体のフローを制御する主プログラムが75枚，合計2,094枚で構成されている．

PASCALコンパイラの調査と検討に約2ケ月，実際のコーディングとデバッグに約2ケ月かかった．デバッグの際見つけた誤りは七ツあり，書き写しの誤りが2，パンチミスが2，FASPサブルーチンのプログラムミスが2，その他のプログラムミスが1という内訳であった．

230－55機によって前記の8 Queenの問題をコンパイルし実行した全時間は約2分かかった．インタプリタのうち，特に使用頻度が高い割に能率の悪い命令のデコードの部分をアセンブラで書きなおした結果，処理時間は1分28秒と大いに圧縮できた．これにより実行時間を早めるにはアセンブラを採用することが有効と思われる．さらに中間言語をインタプリトするのではなく，直接230－55機の命令コードを作り出すコンパイラを作成すれば，さらによい結果が得られるであろう．

本報告でのべたFACOM 230－55用PASCALコンパイラの性能を評価するため，インタプリタに若干のプログラムの追加を行い，所要時間の算定や必要記憶容量が求められるようにした．これによって上述の8 Queenの問題を処理した結果，つぎのようなデータが得られた．

	所要時間（秒）	スタック所要量（語）	ヒープ所要量（語）
コンパイル	17	4,701	1,359
実　　行	66	93	0

10. 謝　　辞

本研究にあたり，Caltechで作成されたIBM用PASCALインタプリタ[9]のコピーを提供して下さった東京大学理学部情報科学科の疋田・安村の両氏に感謝いたします．

デバッグにあたって援助して下さった富士通SE橋本氏，日頃協力して下さる浜田・藤田先生，研究室各位に感謝します．　（1976年4月22日受理）

参 考 文 献

(1) N. Wirth: “The Programming Language Pascal”, Acta Informatica, 1, 35–63, 1971
(2) N. Wirth: “The Programming Language Pascal (Revised Report)”, Berichte der Fachgruppe Computer-Wissenschaften, ETH Zürich, No.5, 1972
(3) N. Wirth: “Systematic Programming”, Prentice Hall, 1973
(4) K. Jensen and N. Wirth: “Pascal User Manual and Report”, Springer, 1974
(5) C. A. R. Hoare and N. Wirth: “Axiomatic Definition of the Programming Language Pascal”, Berichte der Fachgruppe Computer-Wissenschaften, ETH Zürich, No.6, 1972
(6) N. Wirth: “The Design of a PASCAL Compiler”, Software-

Practice and Experience, Vol. 1, 309–333, 1971
(7) U. Ammann: "The Method of Structured Programming applied to the Development of a Compiler", 1973 International Computing Symposium, North-Holland, 1974
(8) K. V. Nori, U. Ammann, K. Jensen and H. H. Nägeli: "The PASCAL 〈P〉 Compiler: Implementation Note", Berichte des Instituts für Informatik, ETH Zürich, No.10, 1974
(9) R. S. Deverill and A. C. Hartmann: "Interpretive PASCAL for the IBM 370", Information Science Technical Report No.6, California Institute of Technology, 1973
(10) 安村・正田；"PASCAL 入門(1)–(6)" 東京大学大型計算機センターニュース，vol. 7，№3－8，1975

付　　録

A　スタック計算機の命令一覧

0	LOD	load contents of address (PQ)
1	LDO	load contents of base level address (Q)
2	STR	store at address (PQ)
3	SRO	store at base level address (Q)
4	LDA	load address (PQ)
5	LAO	load base level address (Q)
6	STO	store
7	LDC	load constant (PQ)
8*	LCI	load constant indirect (Q)
9	IND	indexed fetch (Q)
10	INC	increment address (Q)
11	MST	mark stack (P)
12	CUP	call user procedure (PQ)
13	ENT	enter block (Q)
14	RET	return from block (P)
15	CSP	call standard procedure (Q)
16	IXA	compute indexed address (Q)
17	EQU	compare on equal (PQ)
18	NEQ	compare on not equal (PQ)
19	GEQ	greater than or equal (PQ)
20	GRT	greater than (PQ)
21	LEQ	less than or equal (PQ)
22	LES	less than (PQ)
23	UJP	unconditional jump (Q)
24	FJP	false jump (Q)
25	XJP	indexed jump (Q)
26	CHK	check against upper and lower bounds (Q)
27	EOF	test on end of file
28	ADI	integer addition
29	ADR	real addition
30	SBI	integer subtraction
31	SBR	real subtraction
32	SGS	generate singleton set
33	FLT	float top of the stack
34	FLO	float next to top of the stack
35	TRC	truncation
36	NGI	integer sign inversion
37	NGR	real sign inversion
38	SQI	square integer
39	SQR	square real
40	ABI	absolute value of integer
41	ABR	absolute value of real
42	NOT	boolean "not"
43	AND	boolean "and"
44	IOR	boolean "inclusive or"
45	DIF	set difference
46	INT	set intersection
47	UNI	set union
48	INN	test set membership (in)
49	MOD	modulus
50	ODD	test on odd
51	MPI	integer multiplication
52	MPR	real multiplication
53	DVI	integer division
54	DVR	real division
55	MOV	move (Q)
56	LCA	load address of constant (Q)
57	DEC	decrement address (Q)
58	STP	stop
59	LLD	local load (Q)
60	–	illegal code
61	–	illegal code
62	–	illegal code
63	–	illegal code

B　標準手順 (standard procedure) 一覧

0	GET	get new file element
1	PUT	write new file element
2	SAV	save NEW pointer
3	RST	restore NEW pointer
4	NEW	allocate a NEW dynamic record
5	PAK	pack an array
6	WRS	write a string
7	WRH	write hexadecimal
8	WRI	write integer
9	WRR	write real
10	WRC	write character
11	RDI	read integer
12	RDR	read real
13	RDC	read character
14	SIN	sine
15	COS	cosine
16	EXP	exponential
17	LOG	logarithm
18	SQT	square root
19	ATN	arctan
20	NXP	load next pass
21	INP	init new pass
22	WIF	write intermediate file
23	RIF	read intermediate file
24	BPK	pack fullword
25	SPL	split real into 4 halfword
26	USP	pack 4 halfword into real