A Programming Language and its Implementation for a Mini-computer

Takeshi Kiyono,* Katsuo Ikeda* and Takao Ono*

Abstract

In this report, a new programming language, PL/R, and its implementation in a mini-computer, FACOM-R, are described.

PL/R is a compiler level language designed for mini-computer programming, whose facilities ranged up to the assembly language level; it can handle all of the hardware features of the computer. Accordingly, PL/R is suitable for writing every kind of program including system programs such as I/O control and interrupt processing. The implementation of PL/R is performed by a bootstrapping method, in both PL/I and PL/R, utilizing a large scale computer system.

We believe that this method could be used in other computer systems to implement new computer languages efficiently.

1. Introduction

Mini-computers are more suitable for real time controls than arithmetic computations and it is often necessary for the user to write the these programs or the interrupt processing programs. It is rather improper to write these programs using a compiler level language such as FORTRAN or COBOL. On the other hand, an assembly language is very powerful to use the hardware features efficiently, but it is troublesome in programming and debugging.

After all, compiler level languages which can handle the hardware features as well are more desirable for mini-computers than assembly languages.

We designed a new language, PL/R, for system programming of mini-computer whose language specification , implementation and compiler configuration are described in some detail.

2. Language Specification

Program structure: A program is composed of an external procedure which is,

This paper first appeared in Japanese in Joho－Shori (Journal of the Information Processing Society of Japan), Vol. 15, No. 8 (1974), pp. 595～602.

* Department of Information Science, Kyoto University

generally, composed of a procedure statement, conditional statements, unconditional statements and internal procedures.

Declaration statement: Declaration statement is used to declare variables and arrays which is available in the procedure. It must appeare immediately below the procedure statement. There are two kinds of options, INIT and DEF. INIT option assigns the initial value of variables or arrays and DEF option defines the location in absolute address in the main memory.

Unconditional statement: Assignment, GO TO, CALL,RETURN, I/O, ON, AKI, CM statements and DO unit are treated as unconditional statements. Operaters between level 5 and 10 in the Table 1 may be used in the expression. The accumulater register, ACC, used for the operation may appear explicitly in the expression. Input/Output and peripheral device control or device status check can be expressed explicitly. I/O statement has parameters such as the device address ,the command word,etc.

Operator		Level	
OR	b	1	Logical Or
AND	b	2	Logical And
NOT	u	3	Logical Not
>,<,=	b	4	Comparative
BIT	b	4	Compare Bit "n" with "1"
@	b	5	Exclusive Or
/	b	5	Or
&	b	6	And
#	u	7	Not
+,-	b	8	
-	u	9	
SLL	b	10	Shift Left Logical
SRL	b	10	Shift Right Logical
SLC	b	10	Shift Left Circular
SRA	b	10	Shift Right Arithmetic

b : binary op. u : unary op.

Table. 1 Operators of PL/R

ON, AKI, CM statements are used for the interrupt processing. ON TRAP statement denotes the entry point into the interrupt processing routine and ON RESET statement denotes the return point, if needed, after interrupt processing. AKI statement is used to get the device address which causes the interruption. CM statement changes the mode of operation from normal mode to interrupted mode and vice versa.

Data: The simple variable, the one-dimentional array and the constant can be used in PL/R,which have no attribute. There are four types of constants in expression, decimal constant ,hexadecimal constant,bit constant and character constant.

3. Implementation of PL/R

The implementation of PL/R to FACOM-R is performed by the bootstrapping method using a large scale computer, FACOM 230-75. First, we make a kernel compiler in the large scale computer system using PL/I. Then, the PL/R compiler written in PL/R is compiled by the kernel compiler in the large scale computer system and a object module of the PL/R compiler which runs in the mini-computer is obtained. The sequence of the bootstrapping is shown in Fig. 1 . In the figure, PL/R COMP in PL/I denotes

the PL/R compiler written in PL/I. ML-75 and ML-R denote the machine languages of FACOM 230-75 and FACOM-R.

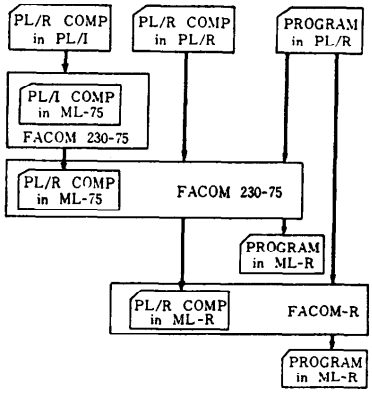


Fig.1 Bootstrapping Method

4. PL/R Compiler Configuration

The PL/R compiler has the following four passes.

4.1 PASS I : Lexical analysis

4.2 PASS II: Syntax analysis

Using a Production Laguage (PL) shown in Fig. 2, we represent the syntax analysis algorithm and construct the syntax analysis routine. It would be more general to make a syntax analysis pass automatically using a interpreter of PL.

PL has production rules (rewriting rules) of the grammer and defines the order to apply them. PL statements are interpreted incorporating a stack memory. If the symbol sequence on the stack matches with the one of the left part of the production rule, it is replaced with the right side of the production rule and the corresponding semantic routine is executed. If it doesn't match, the statement is skipped.

*****	PRODUCTION SYNTAX OF EXPRESSION PART *****					
LABEL	PRODUCTION RULE			ACTION	SCAN	NEXT
TM2		⟨TERM 3⟩ ⟨OP-3⟩			*	EXP0
(#2)	⟨TERM 2⟩ ⟨OP-2⟩	⟨TERM 3⟩ ⟨ANY⟩	–> ⟨TERM 2⟩ ⟨ANY⟩	EXEC 74		TM1
		–⟨TERM 3⟩ ⟩ANY⟩	–> ⟨TERM 2⟩ ⟨ANY⟩	EXEC 77		TM1
		⟨TERM 3⟩ ⟨ANY⟩	–> ⟨TERM2⟩ ⟨ANY⟩	EXEC 75		
TM1		⟨TERM 2⟩ ⟨OP-2⟩			*	EXP0
		‡ ⟨TERM 2⟩ ⟨ANY⟩	–> ⟨TERM 1⟩ ⟨ANY⟩	EXEC 77		TM0
		⟨TERM 2⟩ ⟨ANY⟩	–> ⟨TERM 1⟩ ⟨ANY⟩	EXEC 75		
TM0	⟨TERM 0⟩ &	⟨TERM 1⟩ ⟨ANY⟩	–> ⟨TERM 0⟩ ⟨ANY⟩	EXEC 74		EXP1
		⟨TERM 1⟩ ⟨ANY⟩	–> ⟨TERM 0⟩ ⟨ANY⟩	EXEC 75		
EXP1		⟨TERM 0⟩ &			*	EXP0
(#1)	⟨EXP⟩ ⟨OP-1⟩	⟨TERM 0⟩ ⟨ANY⟩	–> ⟨EXP⟩ ⟨ANY⟩	EXEC 74		EXP2
		⟨TERM 0⟩ ⟨ANY⟩	–> ⟨EXP⟩ ⟨ANY⟩	EXEC 75		
EXP2		⟨EXP⟩ ⟨OP-1⟩			*	EXP0
		⟨EXP⟩ ⟨OP-4⟩			*	EXP0
(#3)		⟨PRIM⟩=⟨EXP⟩;	–> ⟨ASSIGN⟩	EXEC 86		UC0
		⟨VAR⟩ (⟨EXP⟩)	–> ⟨FACT⟩	EXEC 94	*	TM3
		(⟨EXP⟩)	–> ⟨FACT⟩	EXEC 95	*	TM3
		⟨ID⟩ (⟨EXP⟩)	–> ⟨PRIM⟩	EXEC 93	*	PRM0

Fig.2 Production Syntax of PL/R (Expression part)

For example, A=X+(Y/5); has a syntax tree shown in Fig.3. The status of the stack at the stage #1 in Fig.3 is shown in Fig.4 (1) and it is transformed to Fig.4 (2) being applied the production rule,

⟨expression⟩ ⟨operater-1⟩ ⟨term-0⟩ → ⟨expression⟩ .

At the same time, the semantic routine generate a quadruple,

(OR , (VAR, y) , (CONST, 5) , (TEMP, 1)) (1)

(TEMP, 1) is a temporary memory. On the same way, quadruples are generated at the stage #2, #3.

(PLUS , (VAR, x) , (TEMP, 1) , (TEMP, 2)) (2)

(ASSIGN , (TEMP, 2) , (0 ,0) , (VAR, a)) (3)

4.3 PASS III: Optimization and Storage Allocation

Quadruples mechanically generated in PASS II is much redundant. For example, (VAR, x) and (TEMP, 1) in the quadruple (2) are exchangeable and (TEMP, 1)s in the quadruple (1), (2) are redundant. This kind of optimization can be easily excuted by comparing succeeding two quadruples. Quadruples (1), (2), (3) are optimized as follows.

(OR , (VAR, y) , (CONST, 5) , (0 , 0)) (1)'

(PLUS , (0 , 0) , (VAR, x) , (0 , 0)) (2)'

(ASSIGN , (0 , 0) , (0 , 0) , (VAR, a)) (3)'

Storage for variables, arrays, registers, temporary memories, constants and programs are allocated in this pass. The label table which is constructed in PASS II is also completed.

4.4 PASS IV: Code Generation

This pass generates relocatable binary object codes each of which has a control byte for the relocatable loader. An example of a PL/R program and a part of its object code are shown in Fig.5.

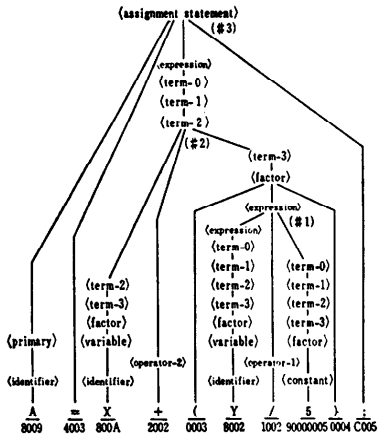


Fig.3 Example of Syntax-tree

(1)

TOS		(SYN)	(SEM)
	10	)	——
	9	⟨term-0⟩	CONST, 5 (5)
	8	⟨operator-1⟩	/
	7	⟨expression⟩	VAR, y (Y)
	6	(	——
	5	⟨operator-2⟩	+
	4	⟨term-2⟩	VAR, x (X)
	3	=	——
	2	⟨primary⟩	VAR, a (A)
	1	⟨procedure head⟩	——
	0	⊢	——

(2)

TOS		(SYN)	(SEM)
	8	)	——
	7	⟨expression⟩	TEMP, 1 (temporary variable)
	6	(	——
	5	⟨operator-2⟩	+
	4	⟨term-2⟩	VAR, x (X)
	3	=	——
	2	⟨primary⟩	VAR, a (A)
	1	⟨procedure head⟩	——
	0	⊢	——

Fig.4 Syntax-analysis Stack

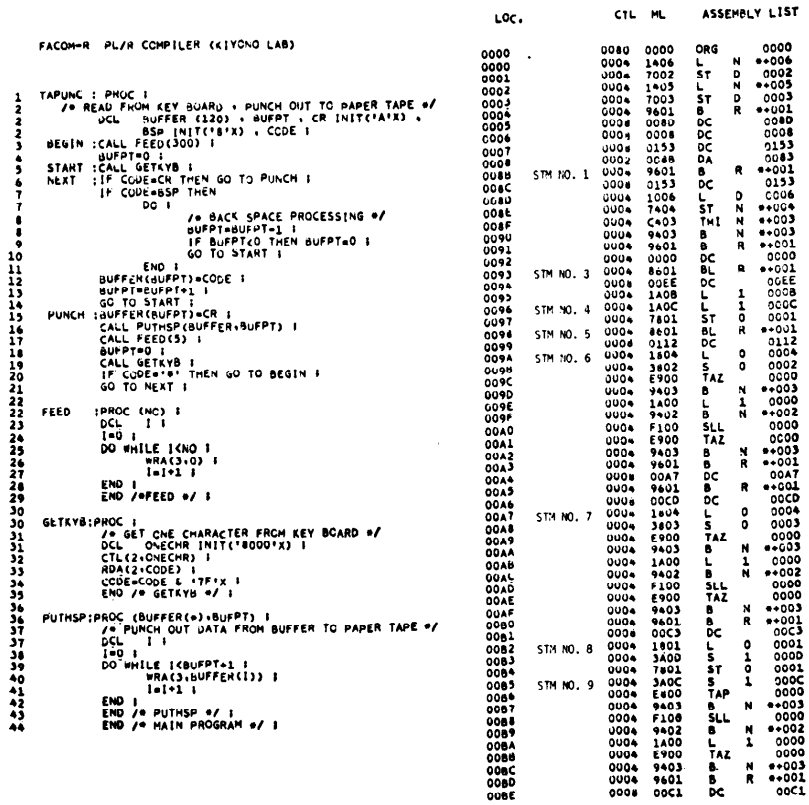
5. Conclusion

The efficiency of a program written in PL/R becomes almost the same as the one written in the assembly language, with the exception of IF statement and DO WHILE statement. In case of address calculation of array, the efficiency may be reduced, because address calculation must be repeated every time even if the same array element is accessed. However, we will be able to compensate the deficiency by optimization. As general logical expression is allowed for conditional term, IF statement and DO WHILE statement generate rather in-efficient object code and th run time efficiency of a PL/R program is affected considerably as bad as one half of an assembly language program at the worst case. Yet, taking into account that mini-computers are often used

in dedicated interactive processing with a single user, we receive more profit of the improvement of the programming efficiency through the use of a compiler language than we pay for the run-time deficiency.

We made a macro-processor using PL/R which was about 600 statements in PL/R and whose object module became 3600 steps. We were able to complete it within much shorter period of a month than it would taken when we had programmed in an assembly language.

We are now revising the PL/R compiler by introducing automatic variables and pointer variables and by improving the facility of interrupt processing.



```
FACOM-R  PL/R COMPILER (KIYONO LAB)

 1  TAPUNC : PROC ;
 2    /* READ FROM KEY BOARD , PUNCH OUT TO PAPER TAPE */
 2         DCL   BUFFER (120) , BUFPT , CR INIT('A'X) ,
 2               BSP INIT('8'X) , CODE ;
 3    BEGIN :CALL FEED(300) ;
 4          BUFPT=0 ;
 5    START :CALL GETKYB ;
 6    NEXT  :IF CODE=CR THEN GO TO PUNCH ;
 7          IF CODE=BSP THEN
 7               DO ;
 8                    /* BACK SPACE PROCESSING */
 8                    BUFPT=BUFPT-1 ;
 9                    IF BUFPT<0 THEN BUFPT=0 ;
10                    GO TO START ;
11               END ;
12          BUFFER(BUFPT)=CODE ;
13          BUFPT=BUFPT+1 ;
14          GO TO START ;
15    PUNCH :BUFFER(BUFPT)=CR ;
16          CALL PUTHSP(BUFFER,BUFPT) ;
17          CALL FEED(5) ;
18          BUFPT=0 ;
19          CALL GETKYB ;
20          IF CODE='0' THEN GO TO BEGIN ;
21          GO TO NEXT ;
22
23    FEED  :PROC (NC) ;
24          DCL   I ;
25          I=0 ;
26          DO WHILE I<NC ;
27               WRA(3,0) ;
28               I=I+1 ;
29          END ;
30          END /*FEED */ ;
31
32    GETKYB:PROC ;
33          /* GET ONE CHARACTER FROM KEY BOARD */
34          DCL   ONECHR INIT('8000'X) ;
35          CTL(2,ONECHR) ;
36          RDA(2,CODE) ;
37          CODE=CODE & '7F'X ;
37          END /* GETKYB */ ;
38
39    PUTHSP:PROC (BUFFER(*),BUFPT) ;
40          /* PUNCH OUT DATA FROM BUFFER TO PAPER TAPE */
41          DCL   I ;
42          I=0 ;
43          DO WHILE I<BUFPT+1 ;
44               WRA(3,BUFFER(I)) ;
41               I=I+1 ;
42          END ;
43          END /* PUTHSP */ ;
44          END /* MAIN PROGRAM */ ;
```

```
LOC.          CTL  ML    ASSEMBLY LIST

                   0080  0000  ORG       0000
0000  .             0004  1A06  L     N   **006
0000                 0004  7002  ST    D   0002
0001                 0004  1A05  L     N   **005
0002                 0004  7003  ST    D   0003
0003                 0004  9601  B     R   **001
0004                 0008  0080  DC        0080
0005                 0004  0008  DC        0008
0006                 0004  0008  DC        0008
0007                 0008  0153  DC        0153
0008                 0002  0C48  DA        0048
0008  STM NO. 1      0004  9601  B     R   **001
008C                 0008  0153  DC        0153
008D                 0004  1006  L     D   0006
008E                 0004  7404  ST    N   **004
008F                 0004  C403  TMI   N   **003
0090                 0004  9403  B     N   **003
0091                 0004  9601  B     R   **001
0092                 0004  0000  DC        0000
0093  STM NO. 3      0004  8601  BL    R   **001
0094                 0008  00EE  DC        00EE
0095                 0004  1A0B  L     1   000B
0096  STM NO. 4      0004  1A0C  L     1   000C
0097                 0004  7801  ST    0   0001
0098  STM NO. 5      0004  8601  BL    R   **001
0099                 0004  0112  DC        0112
009A  STM NO. 6      0004  1A04  L     0   0004
009B                 0004  3802  S     0   0002
009C                 0004  E900  TAZ       0000
009D                 0004  9403  B     N   **003
009E                 0004  1A00  L     1   0000
009F                 0004  9402  B     N   **002
00A0                 0004  F100  SLL       0000
00A1                 0004  E900  TAZ       0000
00A2                 0004  9403  B     N   **003
00A3                 0004  9601  B     R   **001
00A4                 0008  00A7  DC        00A7
00A5                 0004  9601  B     R   **001
00A6                 0008  00CD  DC        00CD
00A7  STM NO. 7      0004  1A04  L     0   0004
00A8                 0004  3803  S     0   0003
00A9                 0004  E900  TAZ       0000
00AA                 0004  9403  B     N   **003
00AB                 0004  1A00  L     1   0000
00AC                 0004  9402  B     N   **002
00AD                 0004  F100  SLL       0000
00AE                 0004  E900  TAZ       0000
00AF                 0004  9403  B     N   **003
00B0                 0004  9601  B     R   **001
00B1                 0008  00C3  DC        00C3
00B2  STM NO. 8      0004  1801  L     0   0001
00B3                 0004  3A00  S     1   000D
00B4                 0004  7801  ST    0   0001
00B5  STM NO. 9      0004  3A0C  S     1   000C
00B6                 0004  E400  TAP       0000
00B7                 0004  9A03  B     N   **003
00B8                 0004  F100  SLL       0000
00B9                 0004  9402  B     N   **002
00BA                 0004  1A00  L     1   0000
00BB                 0004  E900  TAZ       0000
00BC                 0004  9403  B     N   **003
00BD                 0004  9601  B     R   **001
00BE                 0008  00C1  DC        00C1
```

Fig.5 Example of PL/R Source List and Partial List of Object