MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Project MAC

Computation Structures Group Memo 66

Translation of a Block Structured Language Into the Common Base Language

Edward Flinker

This is a thesis presented to the Department of Electrical Engineering for the degree of Bachelor of Science, January 1972

INTRODUCTION

Two of the major problems today's computer user is faced with are:

1. Low level of programming generality (ability of computer system to support modular programming)
2. Difficulties in transferring procedure and data structures from one computer facility to another

One of the ways to solve the above difficulties is by introducing an intermediate representation common to all source languages. Such an intermediate form has been proposed by Prof. J. Dennis of Computation Structures Group (Project MAC, M.I.T.) who calls it Common Base Language. Basic definitions of Common Base Language can be found in References 1 and 2 of this paper. The Base Language is not a block-structured language. I will attempt to simulate block-structured programs written in a source language through translation into the Base Language and then executing it by the Base Language interpreter.

COMMON BASE LANGUAGE

Definition of Common Base Language consists of two stages:

1. Translation of high-level language programs into Common Base Language
2. Interpretation of the program expressed in Common Base Language

ad 1. There will exist a different translator for each high-level language. We may think of Common Base Language as being a machine language and of translator as being a compiler for a given high-level language

ad 2. Interpreter is a nondeterministic state transition system defined by means of giving all states reachable from a given state. A state of the interpreter represents the totality of programs, data and control information present in a computer system (2).

WFPL (What For (?) Programming Language)

```
⟨procedure⟩::=⟨proc⟩⟨body⟩⟨end⟩
⟨proc⟩      ::=⟨idn⟩:PROCEDURE (⟨var_list⟩);|⟨idn⟩:PROCEDURE;
⟨end⟩       ::=END ⟨idn⟩;
⟨var_list⟩  ::=⟨idn⟩|⟨idn⟩,⟨var_list⟩
⟨body⟩      ::=⟨stmnt⟩|⟨stmnt⟩⟨body⟩
⟨stmnt⟩     ::=⟨assign⟩|⟨cond⟩|⟨return⟩|GOTO ⟨idn⟩;|
                ⟨idn⟩:⟨stmnt⟩|⟨dcl⟩|⟨procedure⟩
⟨assign⟩    ::=⟨idn⟩=⟨exp⟩;|⟨idn⟩=⟨idn⟩(⟨var_list⟩);
⟨cond⟩      ::=IF ⟨equal⟩ THEN ⟨stmnt⟩;
⟨equal⟩     ::=⟨exp⟩=⟨exp⟩
⟨exp⟩       ::=⟨lit⟩|⟨idn⟩|⟨exp⟩⟨op⟩⟨exp⟩|(⟨exp⟩)
⟨return⟩    ::=RETURN (⟨idn⟩);
⟨idn⟩       ::=⟨letter⟩|⟨idn⟩⟨letter⟩|⟨idn⟩⟨digit⟩
⟨letter⟩    ::=All capital and lower case letters of the alphabet
⟨lit⟩       ::=⟨digit⟩|⟨lit⟩⟨digit⟩
⟨digit⟩     ::=0|1|2|3|4|5|6|7|8|9
⟨op⟩        ::= +|-|*|/|**
⟨dcl⟩       ::=DECLARE (⟨var_list⟩);
```

For the purpose of this paper I designed a language (WFPL) whose BNF grammar is depicted above. WFPL permits nesting of procedures. Procedures in WFPL return values; they do not return procedure applications. WFPL permits nonlocal references and recursion.

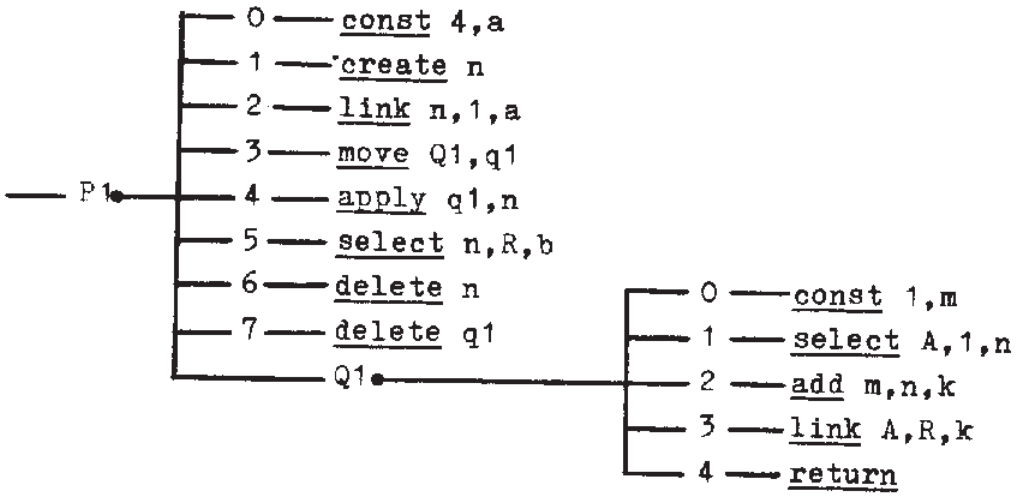
Translation of the following three programs is done on an intuitive rather than a formal basis. The meaning of the presented WFPL programs is also intuitive. The meaning of Base Language programs, however, is defined in terms of the states of the Base Language interpreter. Translation for each program is followed by presenting states of the interpreter during execution. Each of the states is proceeded by a list of Base Language instructions, the execution of which causes this state. For simplification, I present only the data structures since the procedure structure does not change during execution.

PROGRAM 1

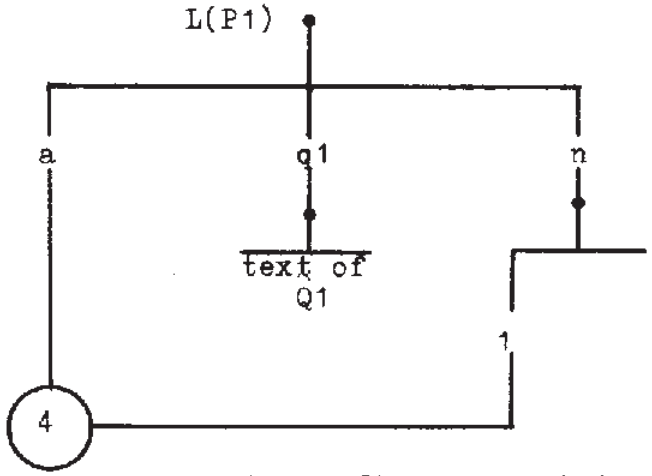
```
 1   P1:PROCEDURE;
 2      DECLARE (a,b,c);
 3      a=4;
 4      b=Q1(a);
 5        Q1:PROCEDURE (n);
 6           DECLARE (m,n,k);
 7           m=1;
 8           k=m+n;
 9           RETURN (k);
10           END Q1;
11      END P1;
```

The program above does not contain non-local references. The problem involves passing of a parameter to procedure Q1. This is done in statement number 4. The other problem is returning of a computed value (k) to calling procedure P1 in statement 9.

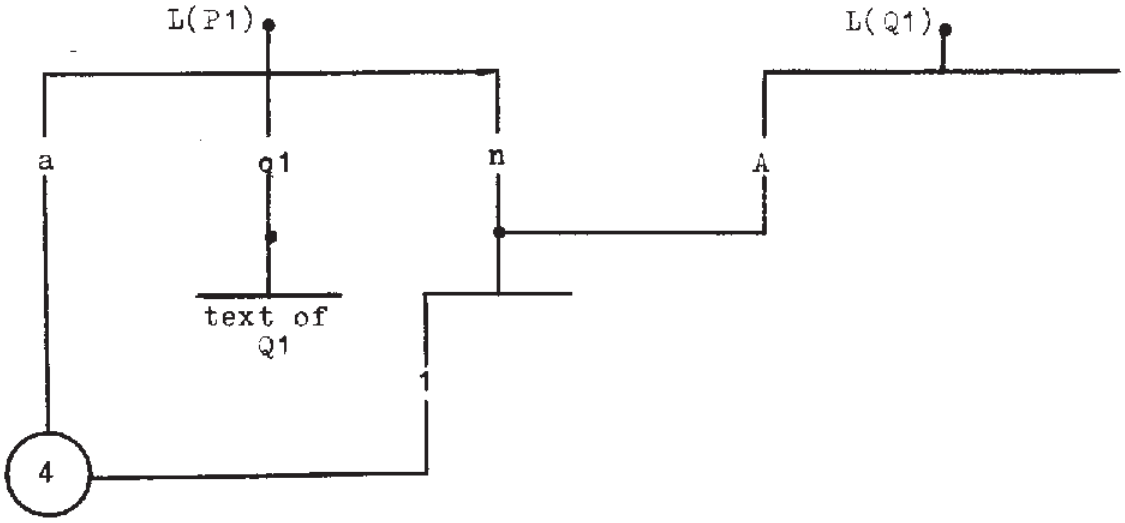
TRANSLATION FOR PROGRAM 1



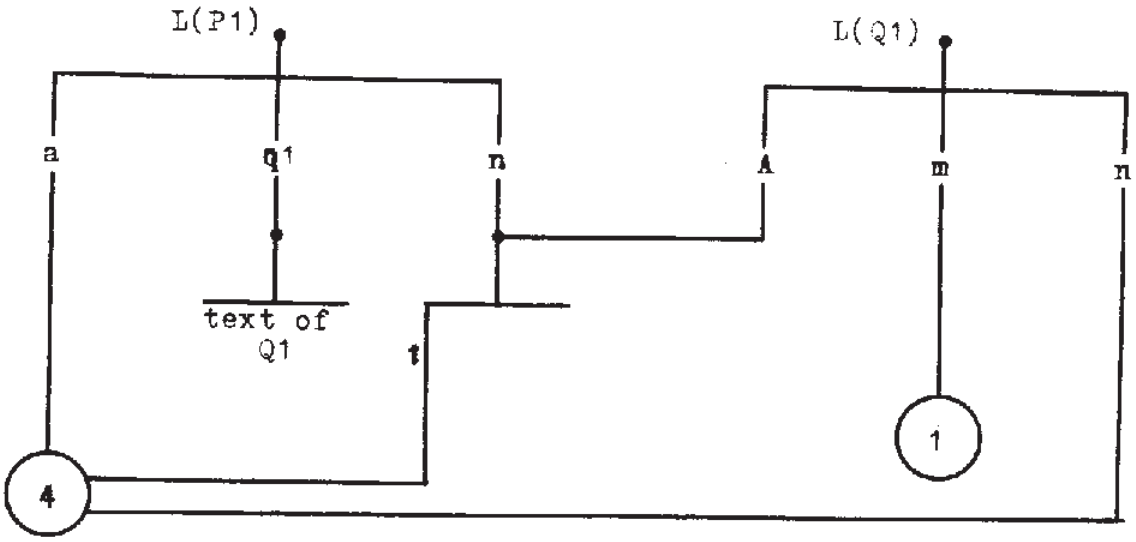
Procedure Structure
for Program 1



The state after: const 4,a ; create n ;
link n,1,a ; move Q1,q1

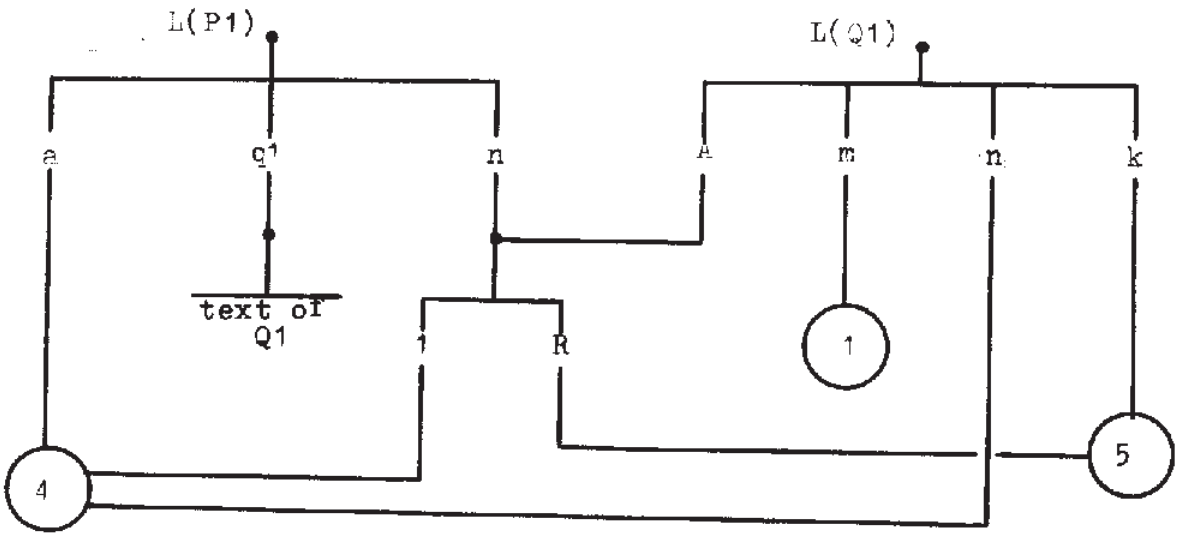


The state after apply q1,n . Execution of this instruction causes instruction select n,R,b in P1 to be made dormant.

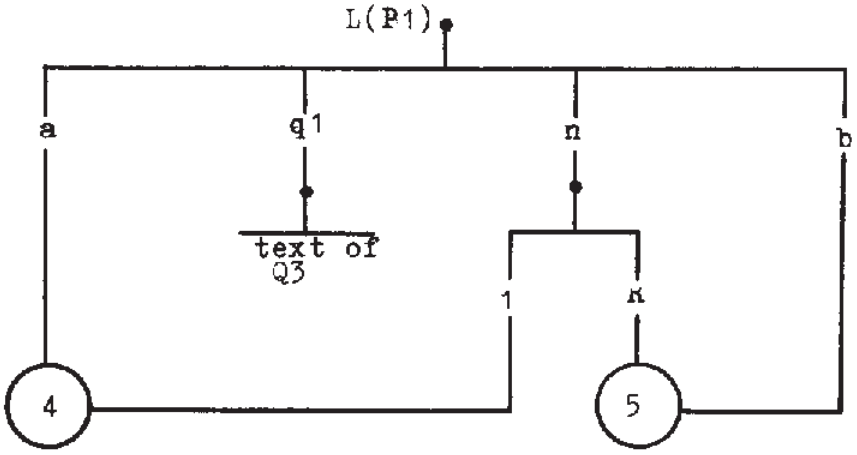


Entering text of Q1.

The state after: const 1,m ; select A,1,n

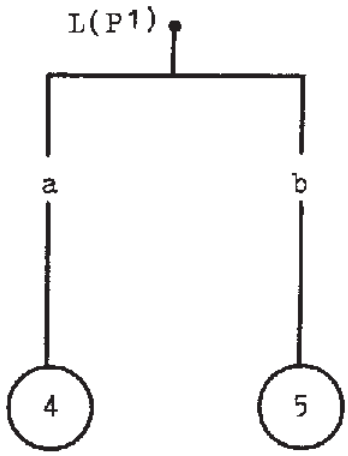


The state after: add m,n,k ; link A,R,k.



Upon return instruction the execution in P1 starts with the dormant instruction.

The state after: return ; select n,R,b .



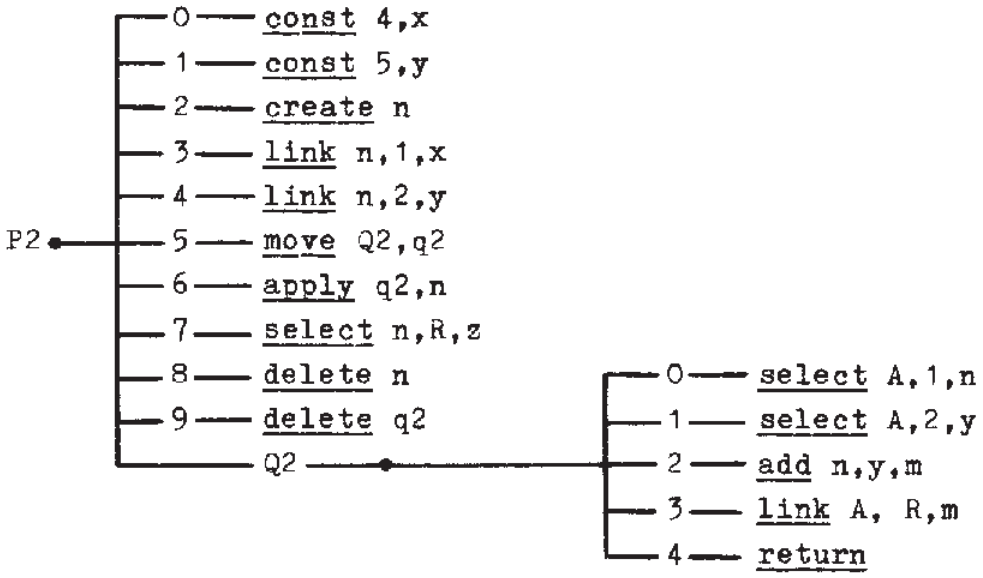
The state after: delete n ; delete q1 .

PROGRAM 2

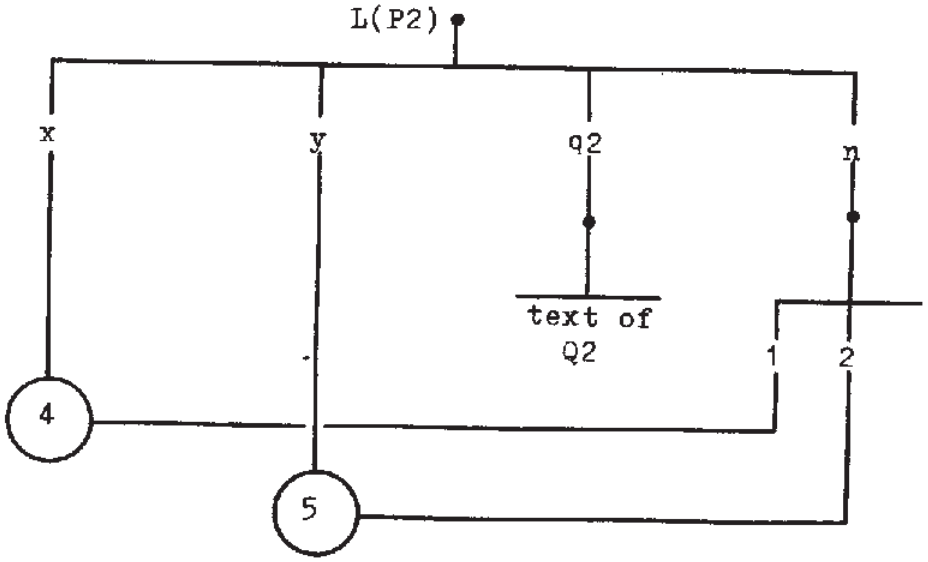
```
1     P2:PROCEDURE;
2        DECLARE (x,y,w,z);
3        x=4;
4        y=5;
5        z=Q2(x);
6          Q2:PROCEDURE (n);
7             DECLARE (m,n);
8             m=n+y;
9             RETURN (m);
10            END Q2;
11       END P2;
```

The program above, in addition to the features of Program 1, contains a nonlocal reference to variable y in statement number 8. The nonlocal variable referenced by a procedure has to be passed to it in a similar fashion as formal parameters of this procedure.

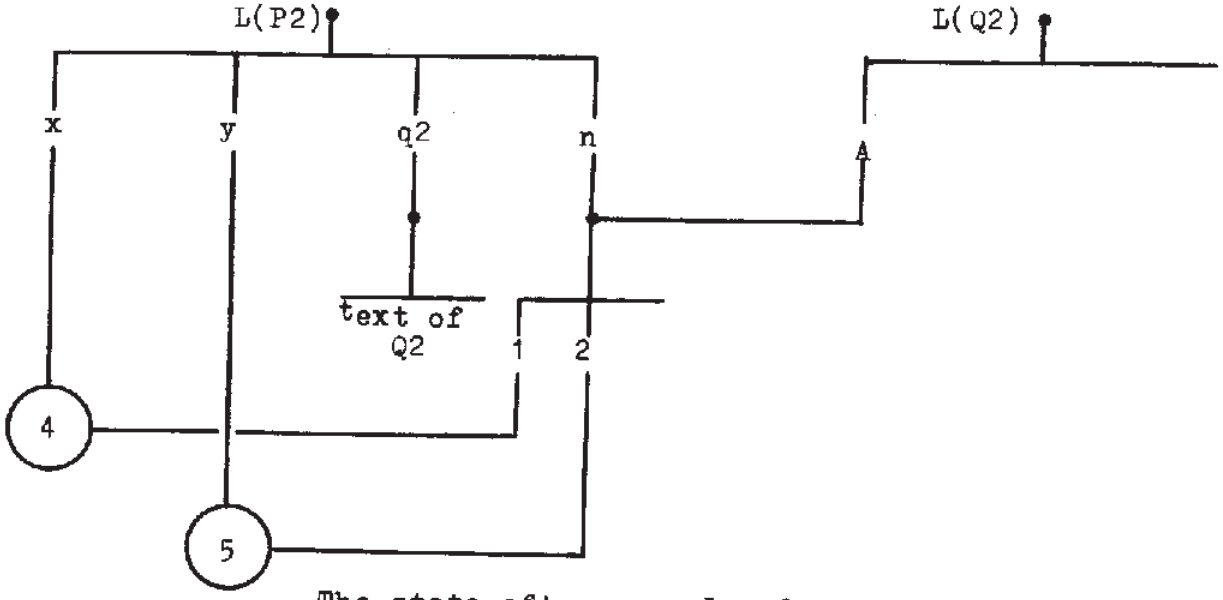
TRANSLATION FOR PROGRAM 2



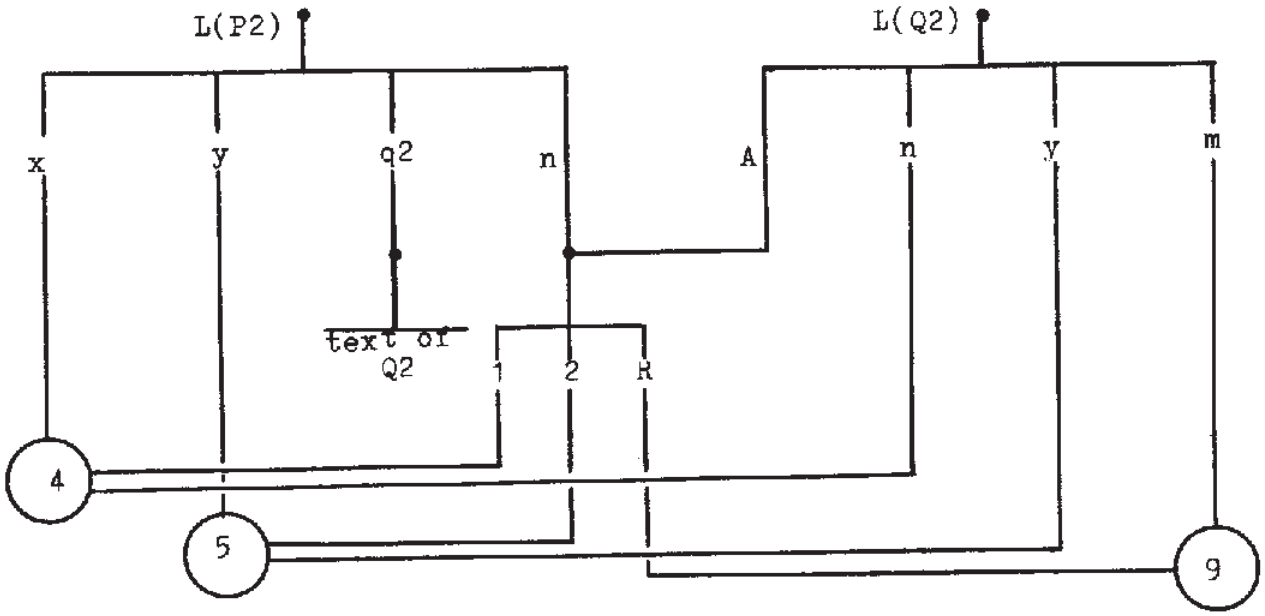
Procedure Structure
for Program 2



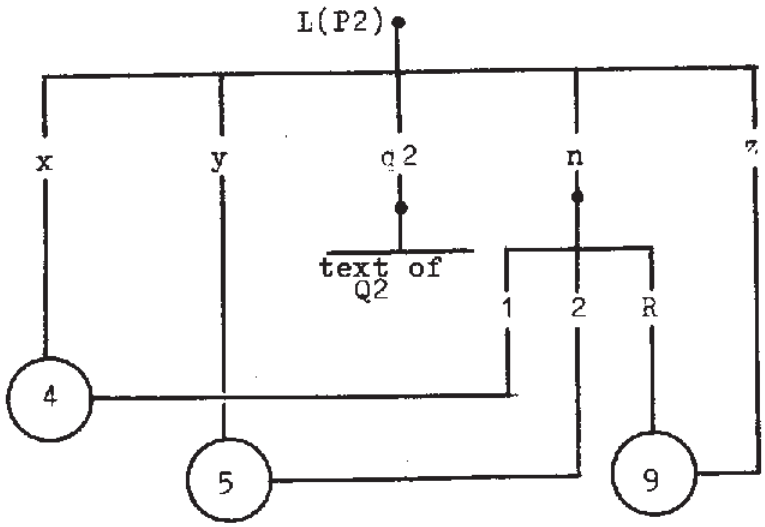
The state after: const 4,x ; const 5,y ; create n ; link n,1,x ; link n,2,y ; move Q2,q2 .



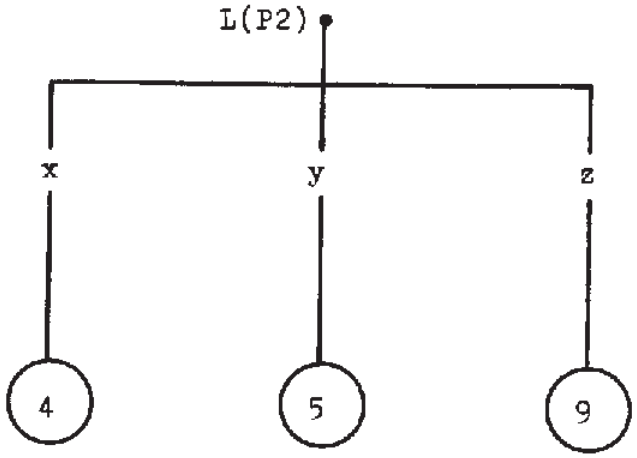
The state after: apply q2,n . Instruction select n,R,z in P2 is made dormant.



Entering text of Q2. The state after: select A,1,n ; select A,2,y ; add n,y,m .



Upon return instruction the execution in P2 starts with the dormant instruction. The state after: return ; select n,R,z .



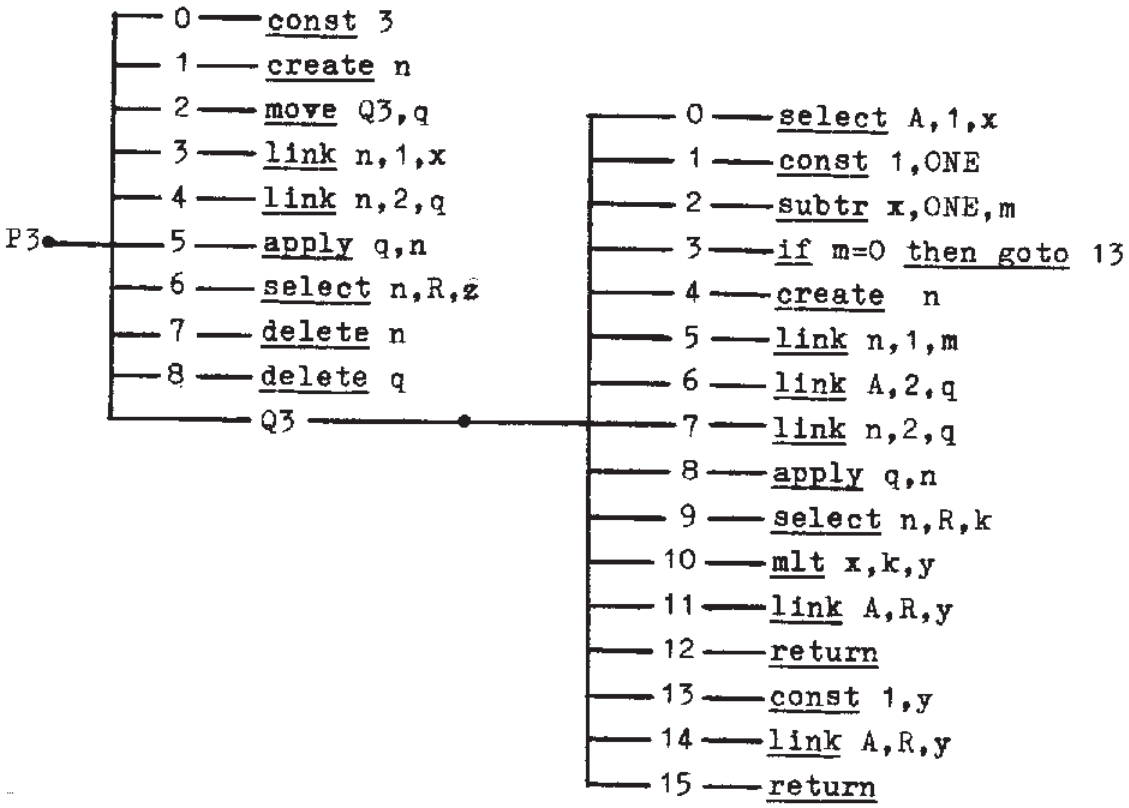
The state after: delete n ; delete q2 .

PROGRAM 3

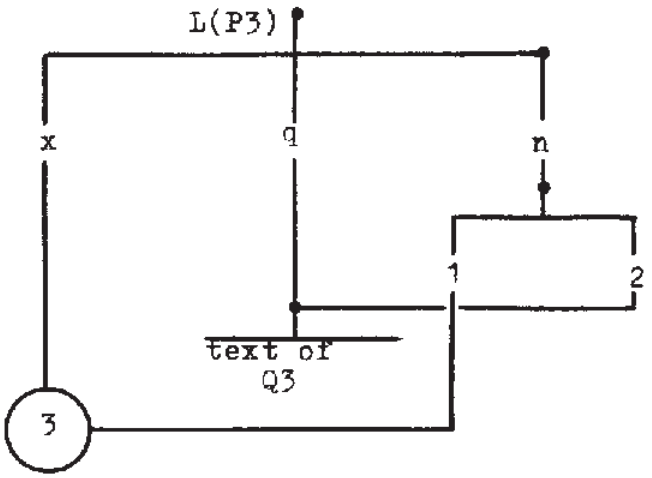
```
 1    P3:PROCEDURE;
 2       DECLARE (x,z);
 3       x=3;
 4       z=Q3(x);
 5         Q3:PROCEDURE (n);
 6            DECLARE (n,m,k,y);
 7            IF n=0 THEN GOTO ALPHA;
 8            m=n-1;
 9            k=Q3(m);
10            y=n*k;
11            GOTO BETA;
12      ALPHA:y=1;
13       BETA:RETURN (y);
14       END Q3;
15       END P3;
```

The program above contains recursive procedure Q3. Procedure Q3 calls itself in order to compute factorial of its parameter.

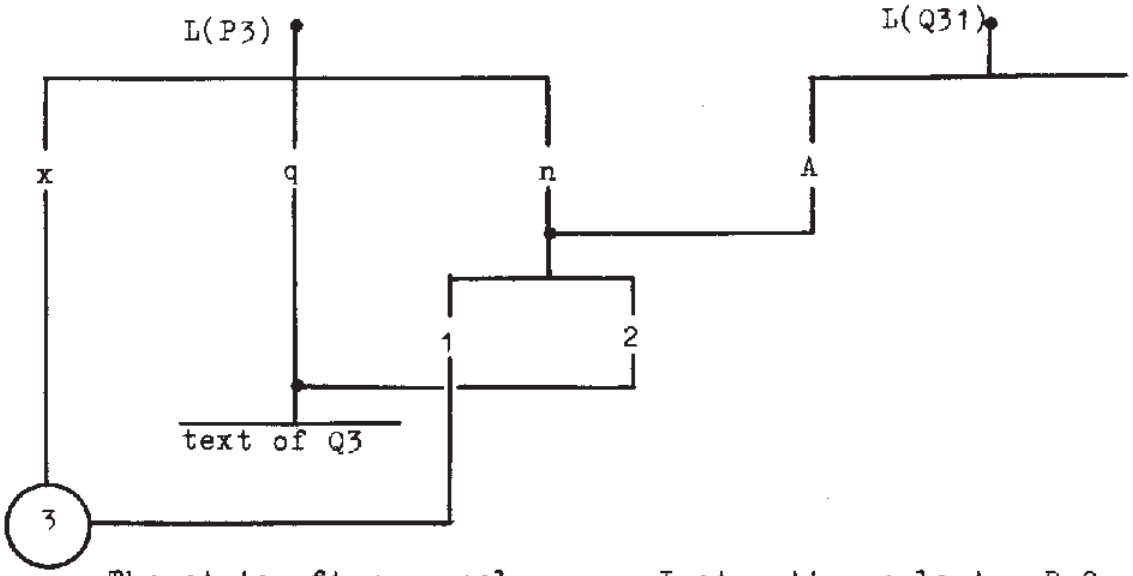
TRANSLATION FOR PROGRAM 3



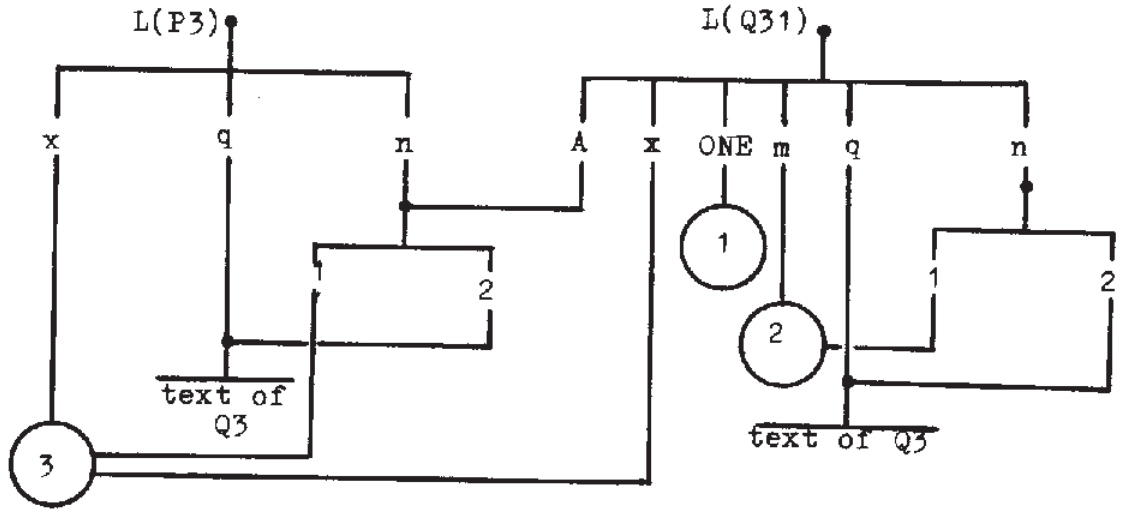
Procedure Structure
for Program 3



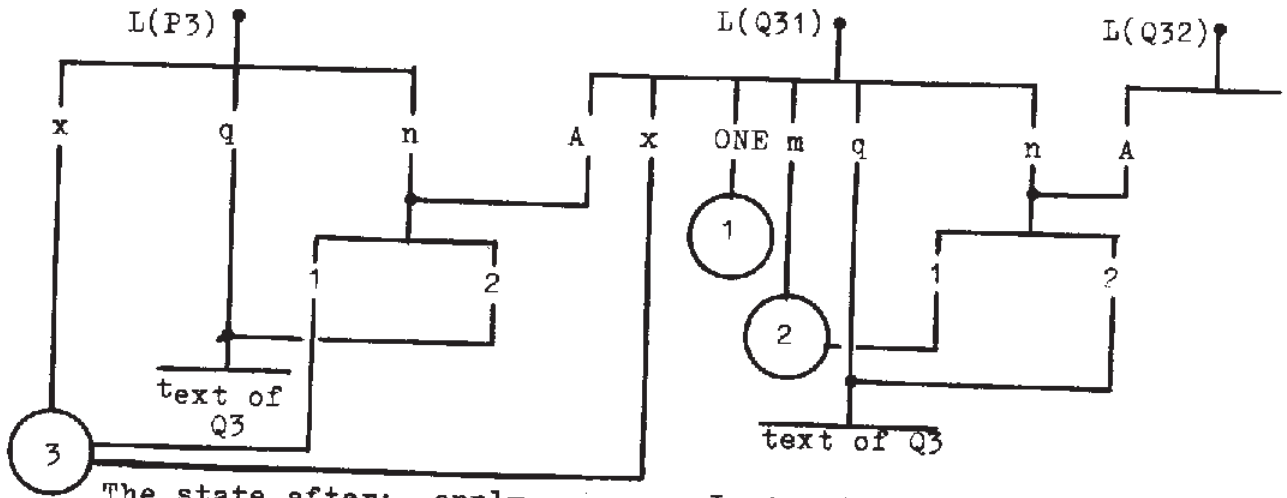
The state after: const 3,x ; create n ; move Q3,q ; link n,1,x ; link n,2,q .



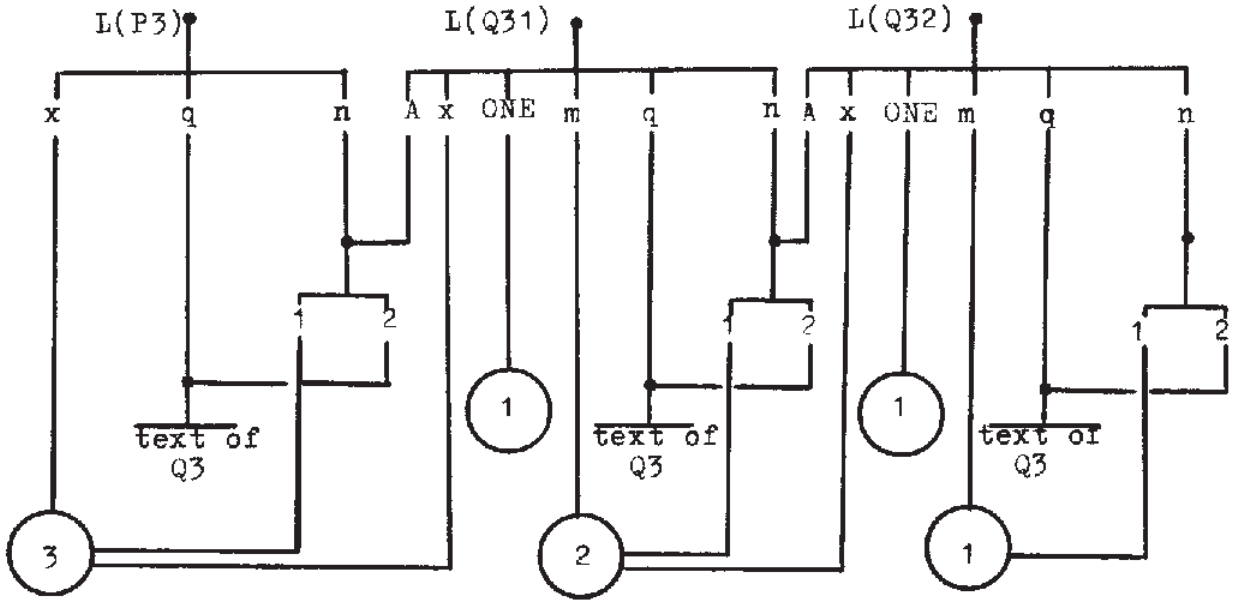
The state after: apply q,n . Instruction select n,R,2 in P3 is made dormant.



Entering the first invocation of Q3. The state after:
select A,1,x ; const 1,ONE ; subtr x,ONE,m ; create n ;
link n,1,m ; link A,2,q ; link n,2,q .



The state after: apply q,n . Instruction select n,R,k is made dormant.

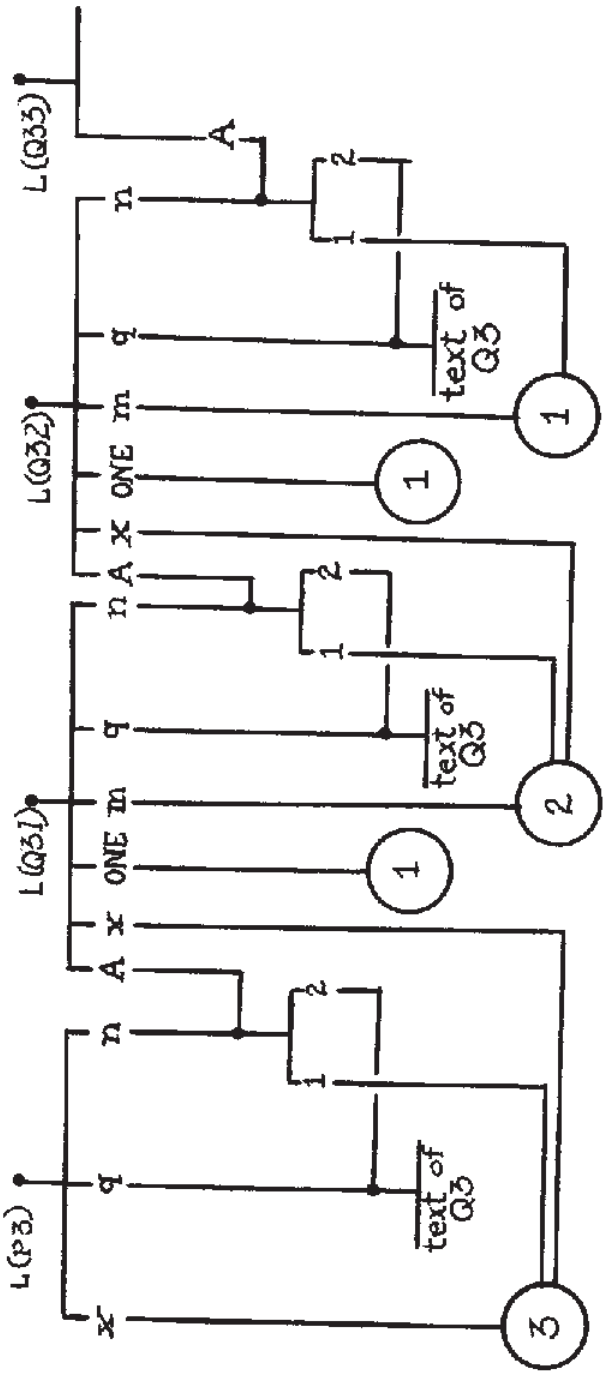


Entering the second invocation of Q3.

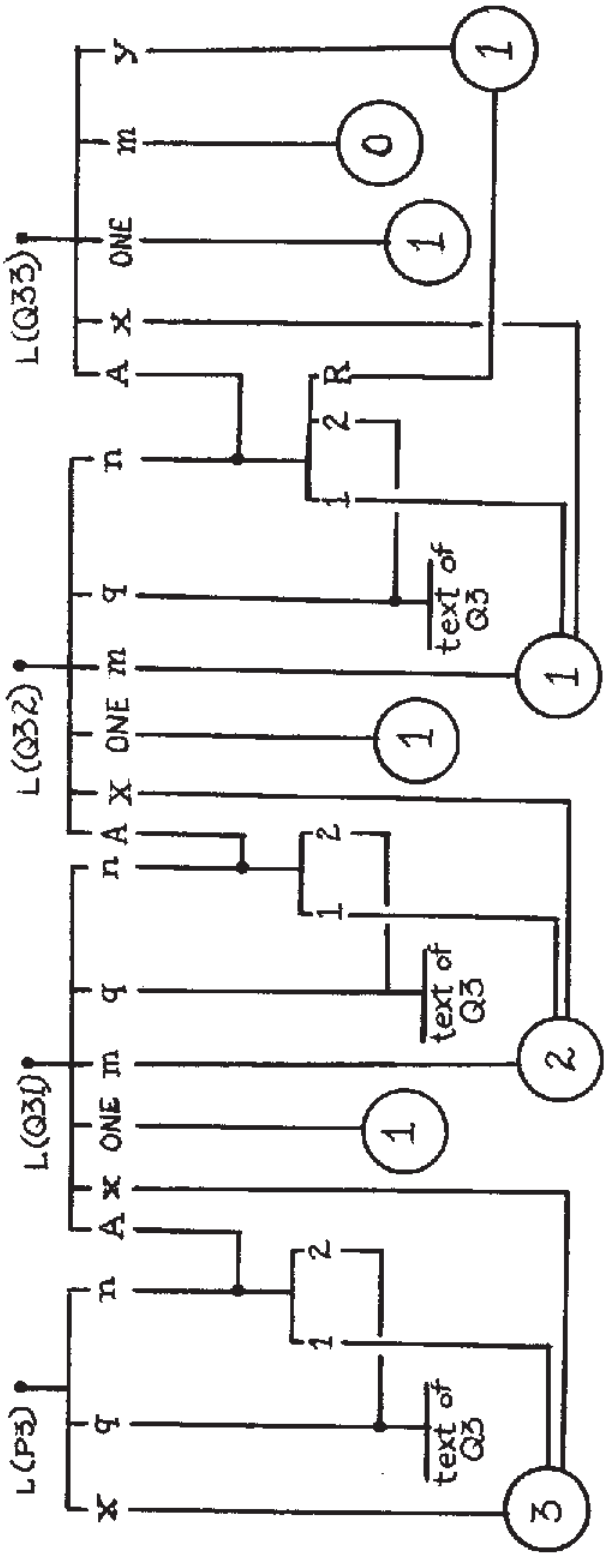
The state after: select A,1,x ; const 1,ONE ; subtr x,ONE,m ; create n ; link n,1,m ; link A,2,q ; link n,2,q .

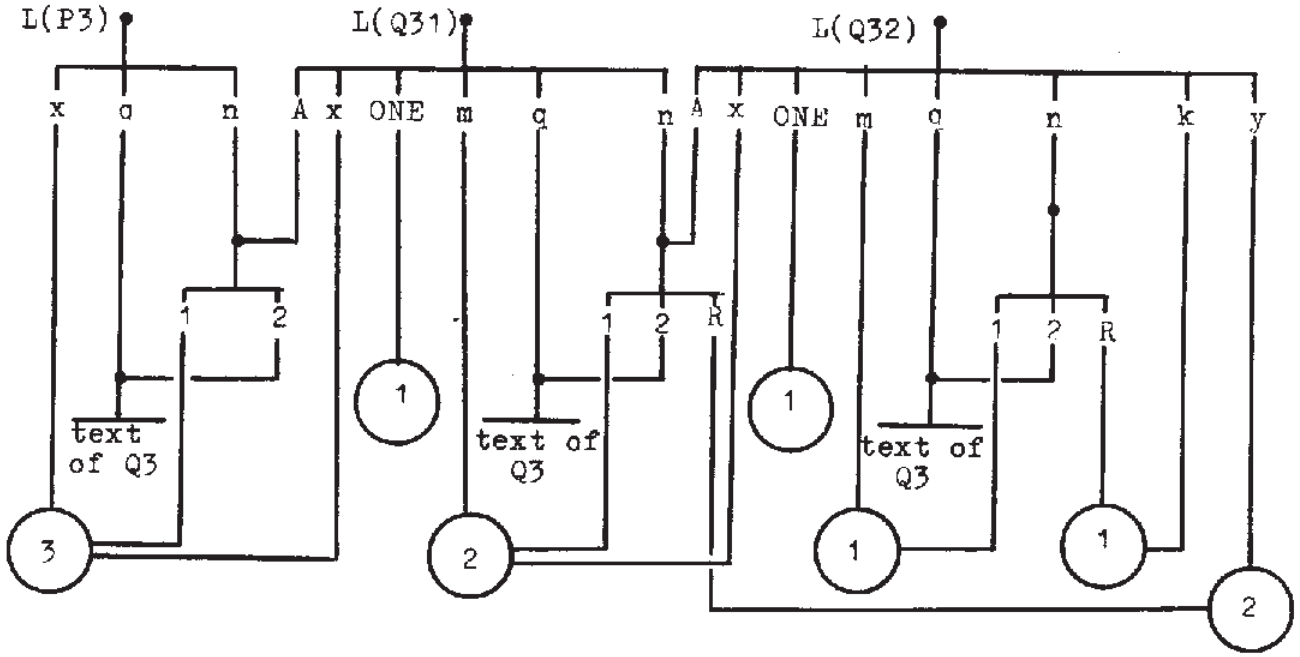
The state after: apply q,n ;

Instruction select n,R,k is made dormant.

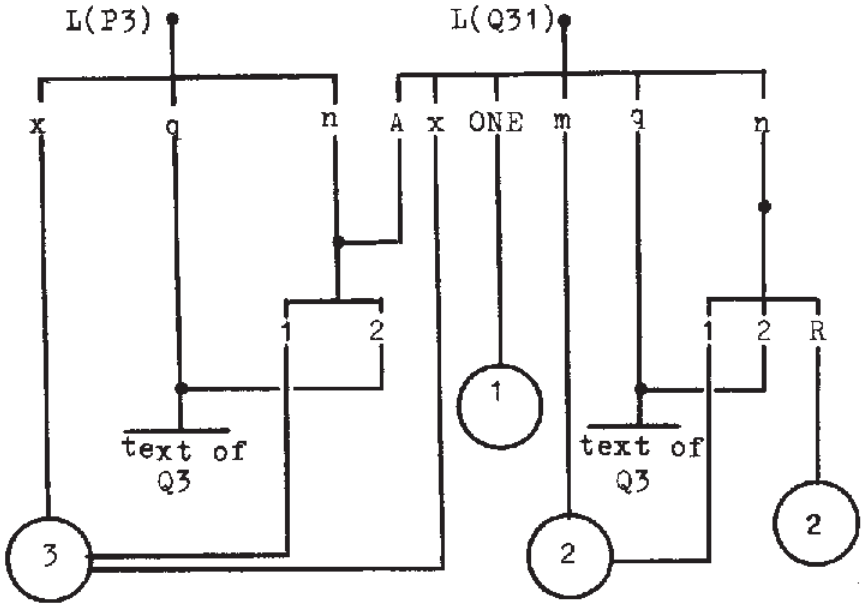


Entering the third invocation of Q3. The state after:
select A,1,x ; const 1,ONE ; subtr x,ONE,m ; const 1,y ;
link A,R,y .

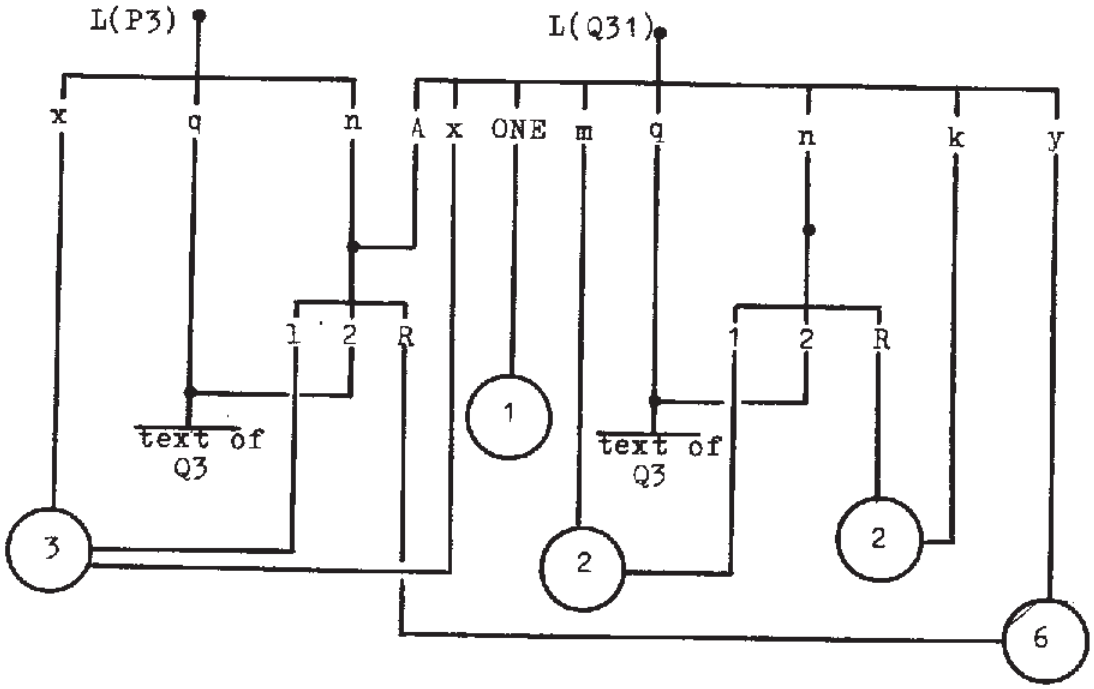




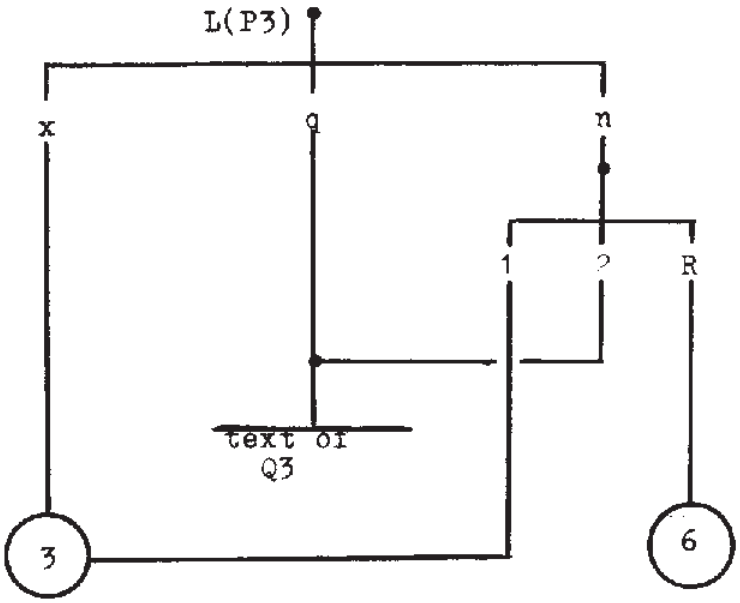
The state after <u>return</u>. We return now from the third to the second invocation of procedure Q3. We continue with the dormant instruction. The state after: <u>select</u> n,R,k ; <u>mlt</u> x,k,y ; <u>link</u> A,R,y .



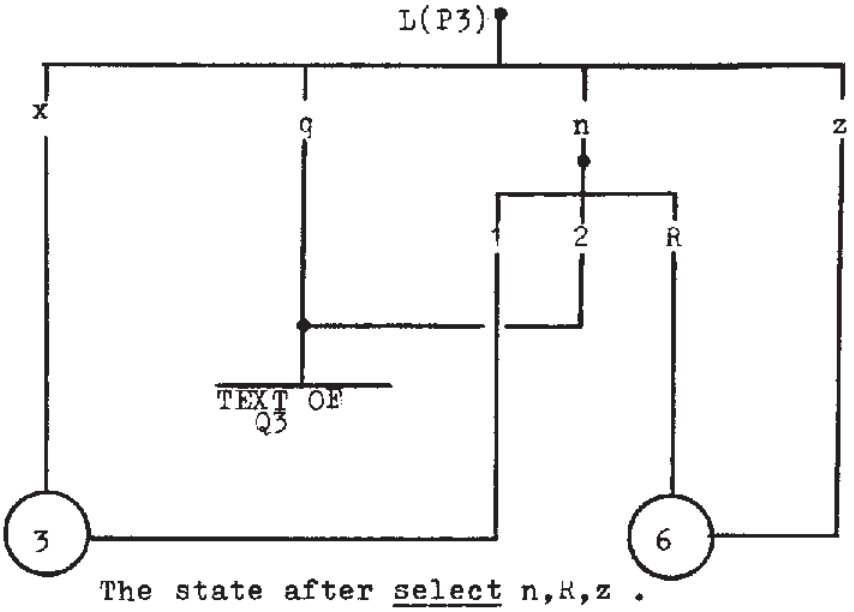
The state after <u>return</u>. We return now from the second to the first invocation of procedure Q3.



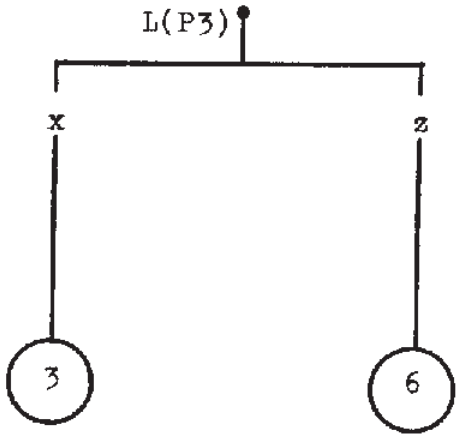
We continue now with the dormant instruction. The state after: select n,R,k ; mlt x,k,y ; link A,R,y .



The state after <u>return</u>. We return now from the first invocation of procedure Q3 to the main procedure P1.



The state after <u>select</u> n,R,z .



We continue in procedure P1 with the dormant instruction. The state after: delete n ; delete q .

As we can see from the presented examples for each invocation of a procedure we have to include in the calling procedure's data structure the following components:

1. Text of the called procedure
2. n-component which contains as its elements parameters passed by the calling procedure, all nonlocal references in the called procedure and text of the called procedure (as in case of recursion).

It is important to realize that in case of recursive call the text of recursive procedure is passed to itself as a parameter. The data structure for a procedure is created on "demand" (automatic storage). A variable is not allocated as long as it has no value assigned to it.

REFERENCES

1. Jack B. Dennis, Programming Generality, Parallelism and Computer Architecture. Computation Structures Group Memo No. 32, Project MAC, M.I.T., Cambridge, Mass., August, 1968.

2. Jack B. Dennis, On the Design and Specification of a Common Base Language. Computation Structures Group Memo No. 60, Project MAC, M.I.T., Cambridge, Mass., July, 1971.