SUBROTINAS

Baseado no Capítulo 6 de Programming Language Processors in Java, de Watt & Brown

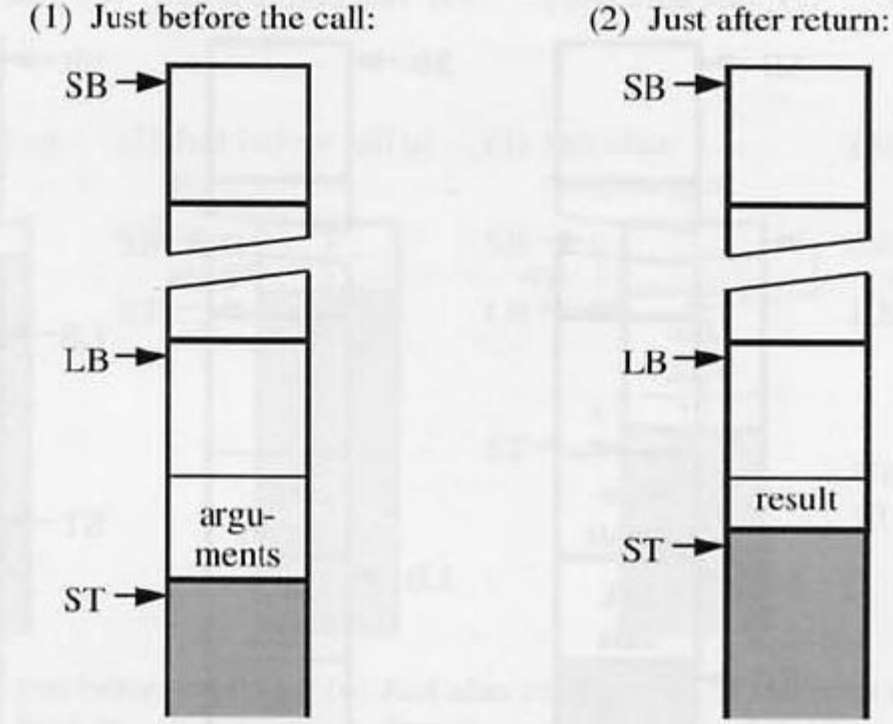
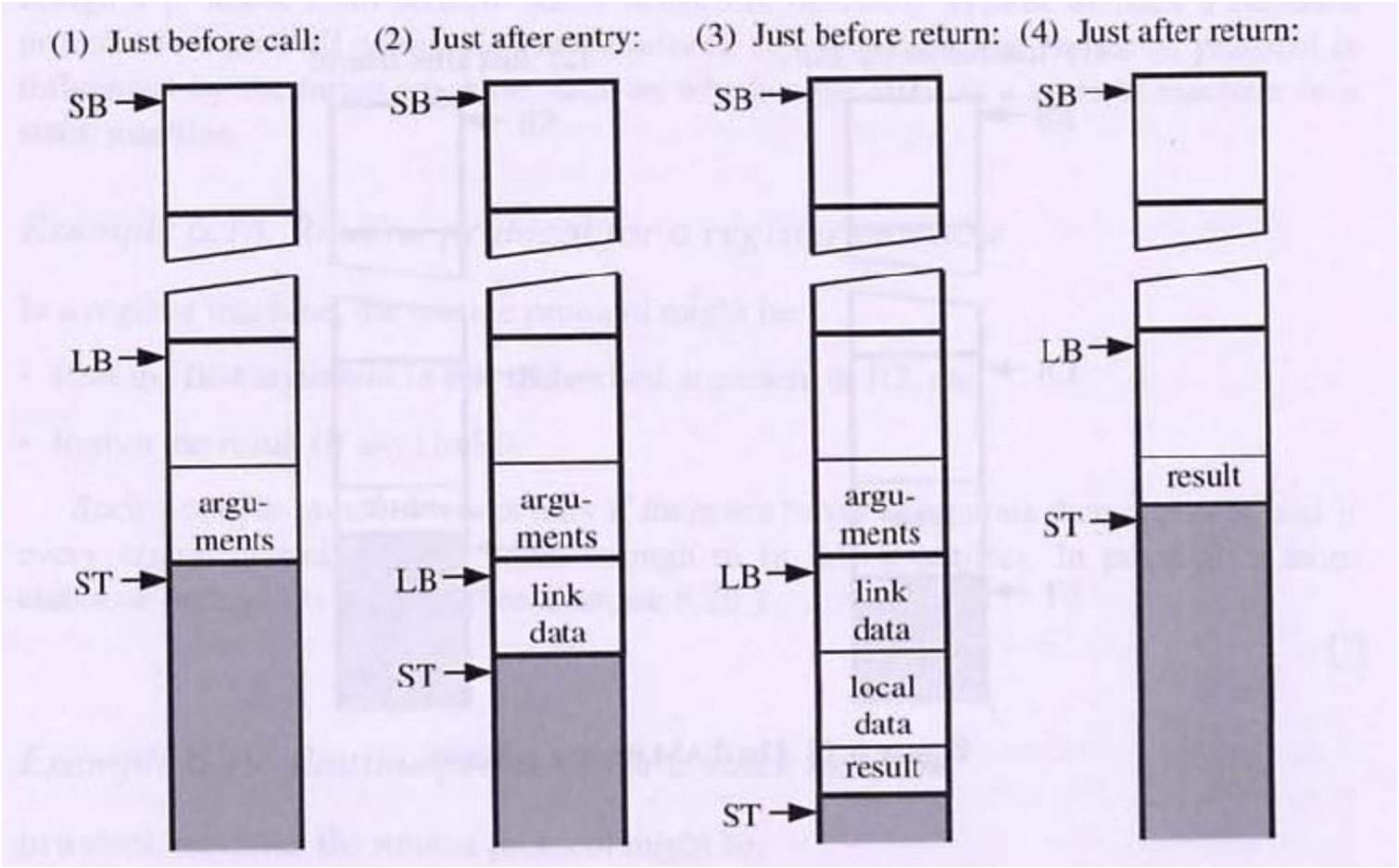
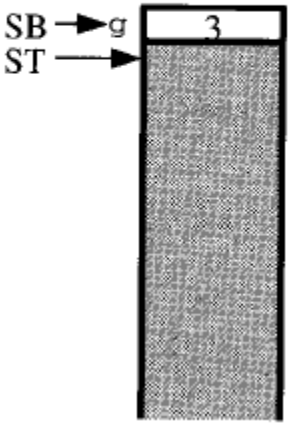


Figure 6.18 The TAM routine protocol.

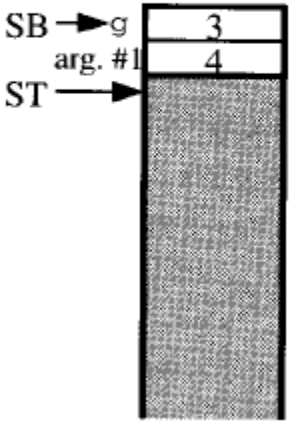

(1) Just before call:
SB
LB
argu-
ments
ST
(2) Just after entry:
SB
argu-
ments
LB
link
data
ST
(3) Just before return:
SB
argu-
ments
LB
link
data
local
data
result
ST
(4) Just after return:
SB
LB
result
ST

```
let var g: Integer;
    func F (m: Integer, n: Integer) : Integer ~
        m * n;
    proc W (i: Integer) ~
        let const s ~ i * i
        in
            begin
            putint(F(i, s));
            putint(F(s, s))
            end
in
    begin
    getint(var g);
    W(g+1)
    end
```

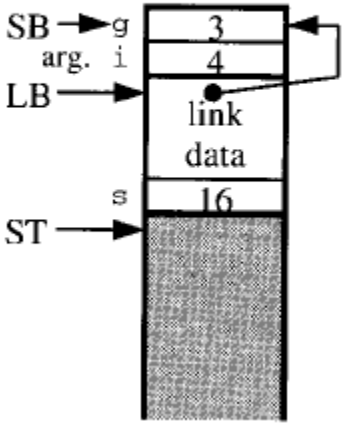
(1) Just after reading g:



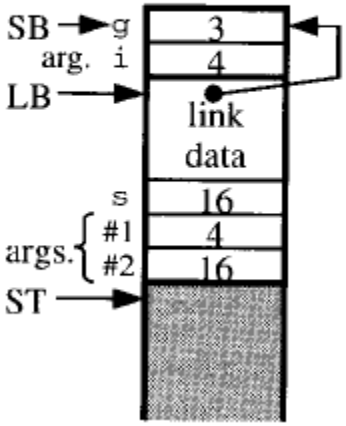
(2) Just before call to W:

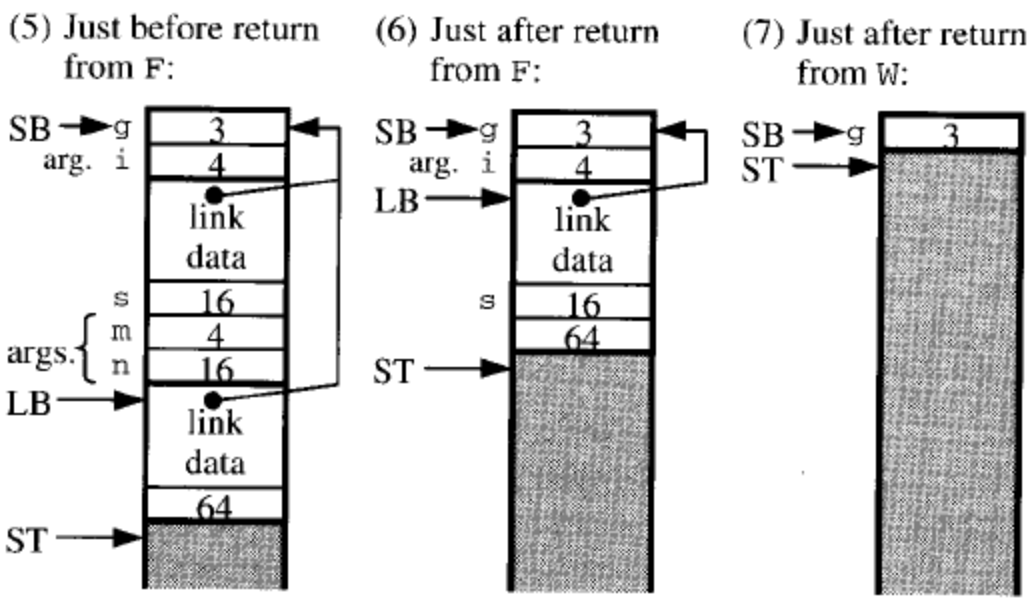


(3) Just after computing s:



(4) Just before call to F:




(5) Just before return from F:
SB
g
arg.
i
3
4
link
data
s
m
n
args.
16
4
16
LB
link
data
64
ST
(6) Just after return from F:
SB
g
arg.
i
3
4
LB
link
data
s
16
64
ST
(7) Just after return from W:
SB
g
ST
3

```
PUSH        1          – expand globals to make space for g
LOADA       0[SB]      – push the address of g
CALL        getint     – read an integer into g
LOAD        0[SB]      – push the value of g
CALL        succ       – add 1
CALL(SB)    W          – call W (using SB as static link)
POP         1          – contract globals
HALT
```

```
W:  LOAD        -1[LB]    – push the value of i
    LOAD        -1[LB]    – push the value of i
    CALL        mult      – multiply; the result will be the value of s
    LOAD        -1[LB]    – push the value of i
    LOAD        3[LB]     – push the value of s
    CALL(SB)    F         – call F (using SB as static link)
    CALL        putint    – write the value of F(i,s)
    LOAD        3[LB]     – push the value of s
    LOAD        3[LB]     – push the value of s
    CALL(SB)    F         – call F (using SB as static link)
    CALL        putint    – write the value of F(s,s)
    RETURN(0)   1         – return, replacing the 1-word argument
                              by a 0-word 'result'
```

```
F:  LOAD        -2[LB]    – push the value of m
    LOAD        -1[LB]    – push the value of n
    CALL        mult      – multiply
    RETURN(1)   2         – return, replacing the 2-word argument pair
                              by a 1-word result
```

Link Estático

```
CALL(SB)  P     – for any call to P

CALL(LB)  Q     – for P to call Q
CALL(L1)  Q     – for Q to call Q (recursively)
CALL(L2)  Q     – for R to call Q
CALL(L1)  Q     – for S to call Q

CALL(LB)  R     – for Q to call R
CALL(L1)  R     – for R to call R (recursively)

CALL(LB)  S     – for P to call S
CALL(L1)  S     – for Q to call S
CALL(L2)  S     – for R to call S
CALL(L1)  S     – for S to call S (recursively)
```

Let R be a routine declared at routine level l (thus the *body* of R is at level l+1). Then R is called as follows:

If $l = 0$ (i.e., R is a global routine):

```
CALL(SB)  R        – for any call to R
```

If $l > 0$ (i.e., R is enclosed by another routine):

```
CALL(LB)  R        – for code at level l to call R
CALL(L1)  R        – for code at level l+1 to call R
CALL(L2)  R        – for code at level l+2 to call R
...
```

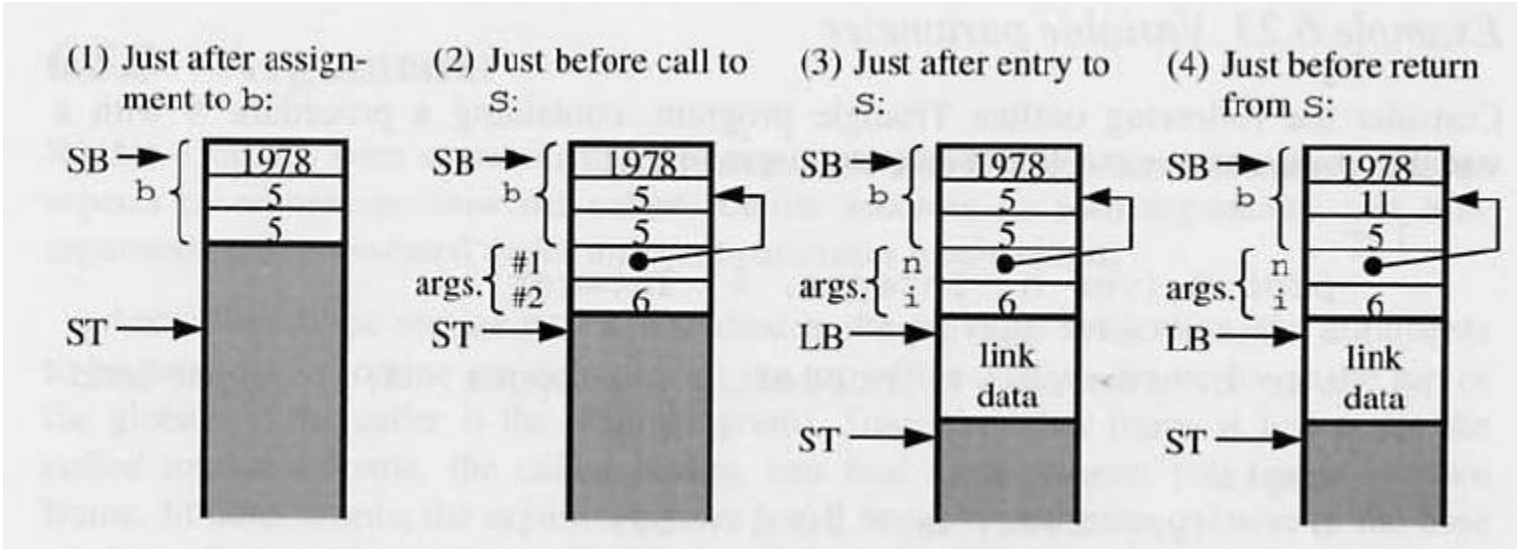
Argumentos

```
LOAD -1[LB]     – for procedure W to fetch its argument i

LOAD -2[LB]     – for function F to fetch its argument m
LOAD -1[LB]     – for function F to fetch its argument n
```

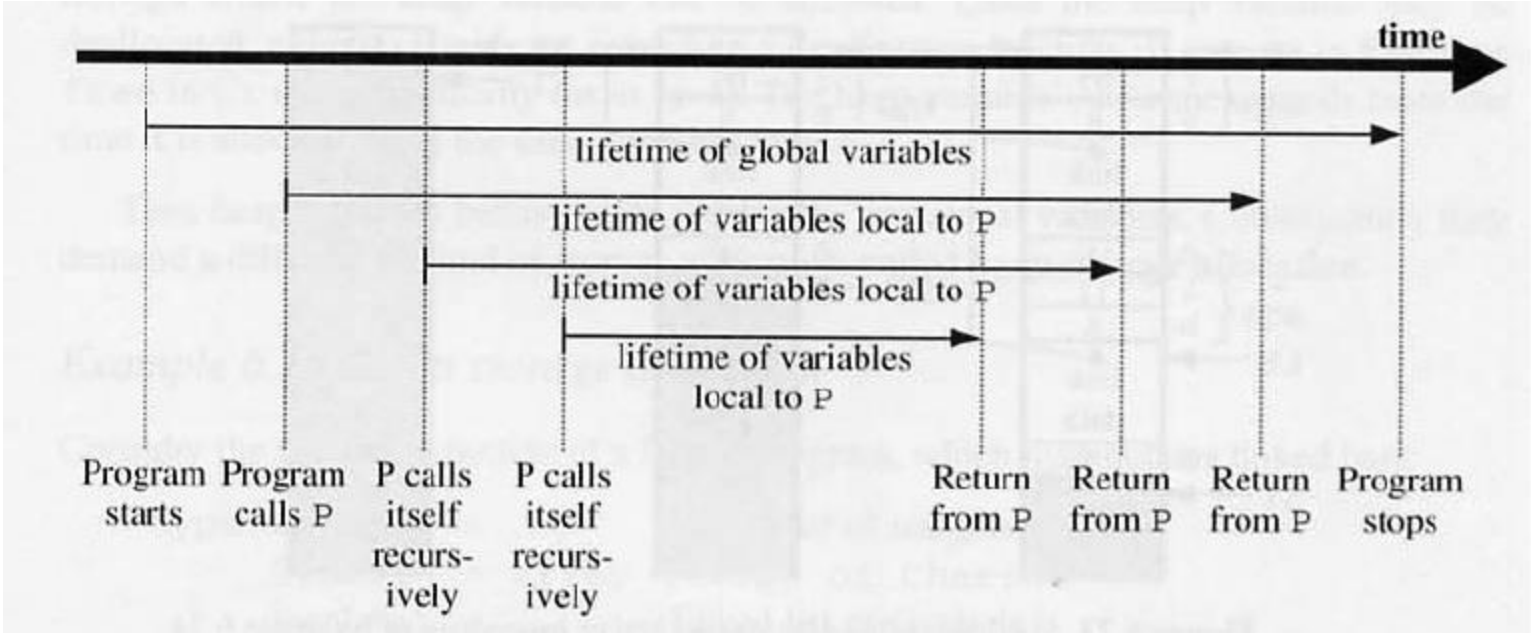
```
let
    proc S (var n: Integer, i: Integer) ~
       n := n + i;
    var b: record y: Integer, m: Integer, d: Integer end
in
    begin
    b := {y ~ 1978, m ~ 5, d ~ 5};
    S(var b.m, 6);
    end
```

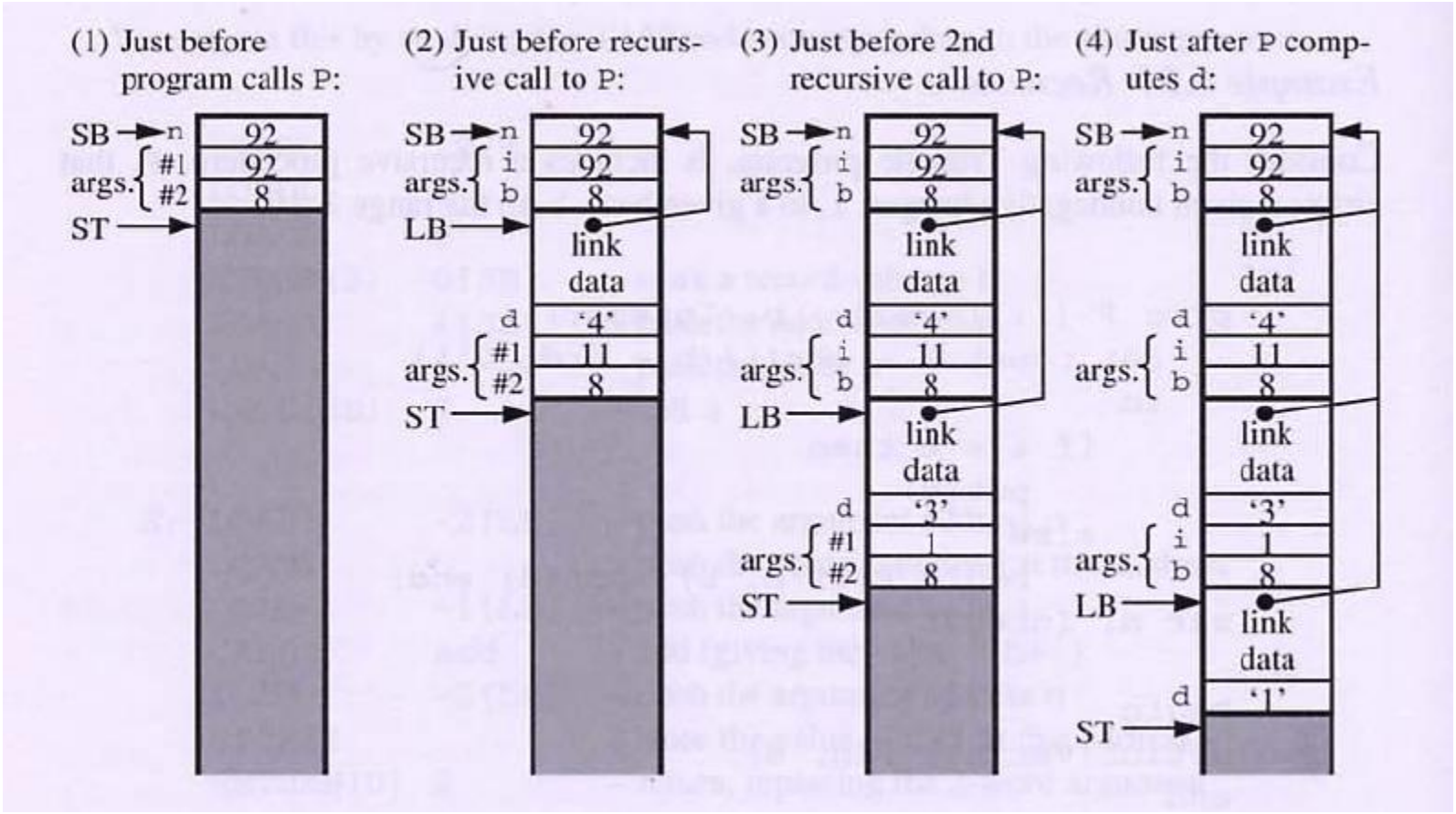
```
        ...
        LOADL     1978
        LOADL     5
        LOADL     5
        STORE(3)  0[SB]    – store a record value in b
        LOADA     1[SB]    – push the address of b.m
        LOADL     6        – push the value 6
        CALL(SB)  S        – call S
        ...
S:      LOAD      -2[LB]   – push the argument address n
        LOADI              – push the value contained at that address
        LOAD      -1[LB]   – push the argument value i
        CALL      add      – add (giving the value of n+i)
        LOAD      -2[LB]   – push the argument address n
        STOREI             – store the value of n+i at that address
        RETURN(0) 2        – return, replacing the 2-word argument
                             pair by a 0-word 'result'
```

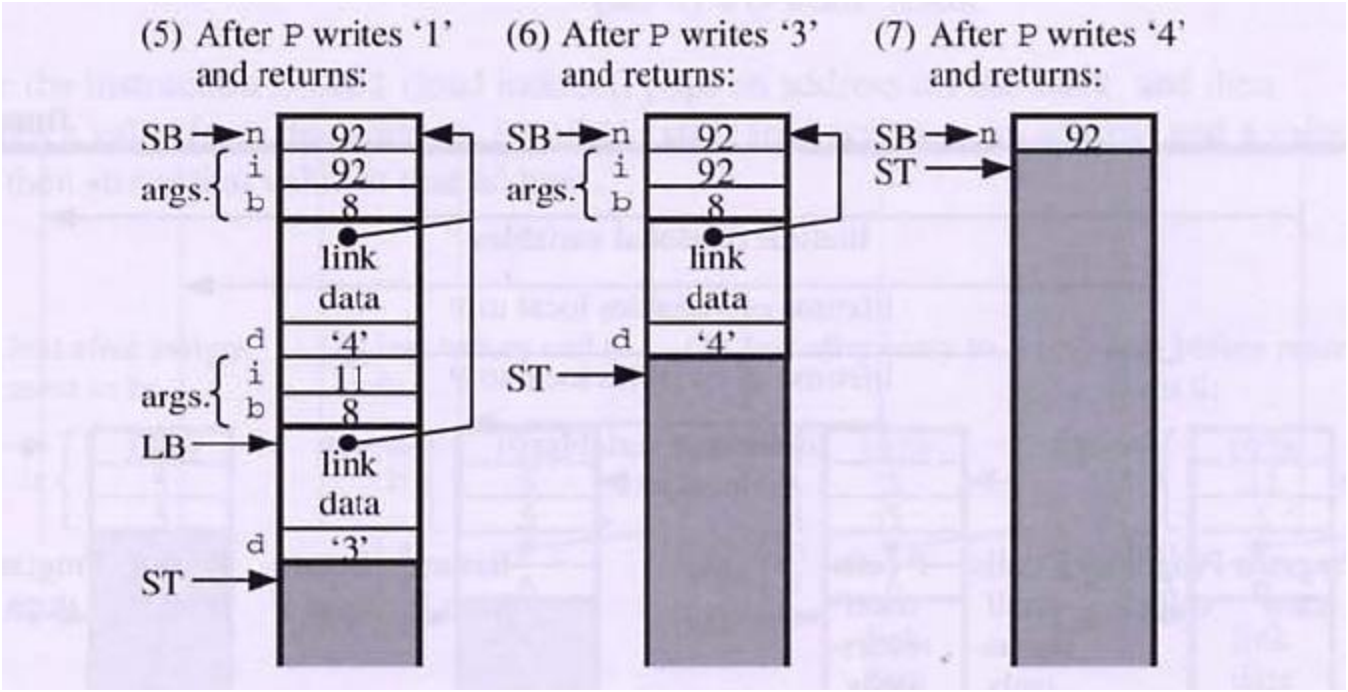

(1) Just after assignment to b:
SB
b
1978
5
5
ST
(2) Just before call to S:
SB
b
1978
5
5
args.
#1
#2
6
ST
(3) Just after entry to S:
SB
b
1978
5
5
args.
n
i
6
LB
link
data
ST
(4) Just before return from S:
SB
b
1978
11
5
args.
n
i
6
LB
link
data
ST

Recursão

```
let
    proc P (i: Integer, b: Integer) ~
        let const d ~ chr(i//b + ord('0'))
        in
            if i < b then
                put(d)
            else
                begin P(i//b, b); put(d) end;
    var n: Integer
in
    begin
    getint(var n); P(n, 8)
    end
```


time
lifetime of global variables
lifetime of variables local to P
lifetime of variables local to P
lifetime of variables local to P
Program starts
Program calls P
P calls itself recurs-ively
P calls itself recurs-ively
Return from P
Return from P
Return from P
Program stops


(1) Just before program calls P:
SB
n
92
args.
#1
92
#2
8
ST
(2) Just before recursive call to P:
SB
n
92
args.
i
92
b
8
LB
link
data
d
'4'
args.
#1
11
#2
8
ST
(3) Just before 2nd recursive call to P:
SB
n
92
args.
i
92
b
8
link
data
d
'4'
args.
i
11
b
8
LB
link
data
d
'3'
args.
#1
1
#2
8
ST
(4) Just after P computes d:
SB
n
92
args.
i
92
b
8
link
data
d
'4'
args.
i
11
b
8
link
data
d
'3'
args.
i
1
b
8
LB
link
data
d
'1'
ST


(5) After P writes '1' and returns:
SB
n
92
args.
i
92
b
8
link
data
d
'4'
i
11
b
8
LB
link
data
d
'3'
ST
(6) After P writes '3' and returns:
SB
n
92
args.
i
92
b
8
link
data
d
'4'
ST
(7) After P writes '4' and returns:
SB
n
92
ST