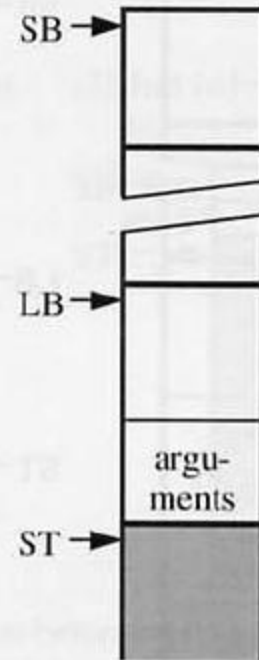


SUBROTINAS

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

Última modificação em 20/05/2026 18:32:30

(1) Just before the call:



(2) Just after return:

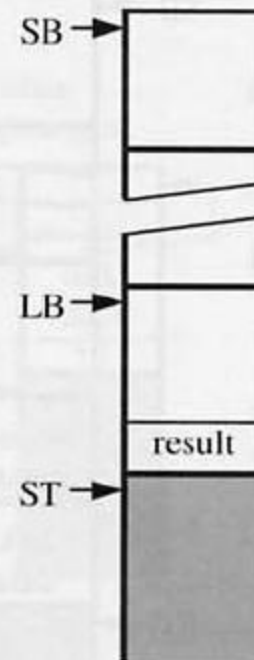
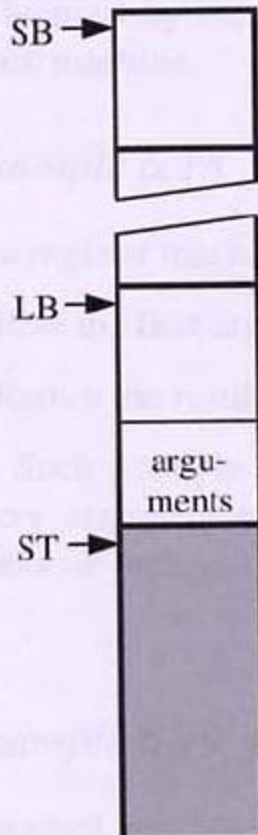
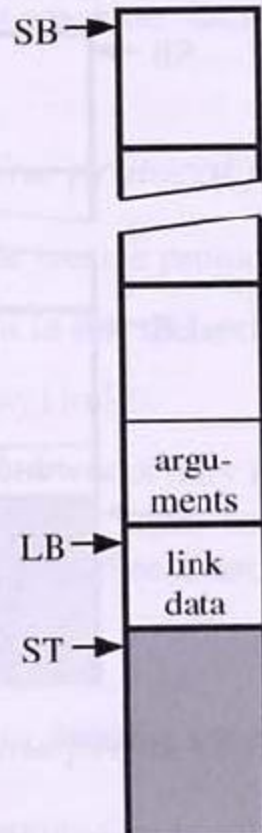


Figure 6.18 The TAM routine protocol.

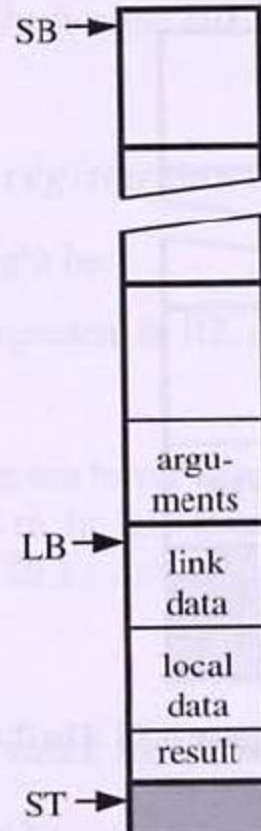
(1) Just before call:



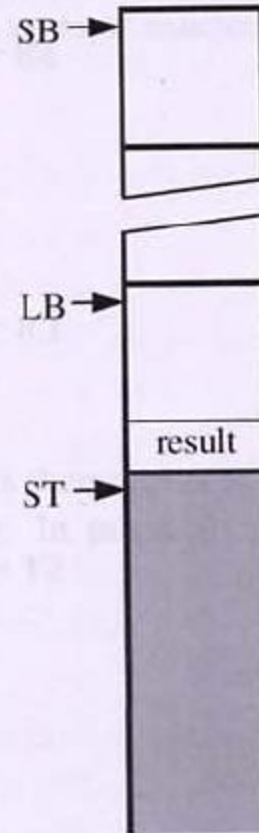
(2) Just after entry:



(3) Just before return:



(4) Just after return:



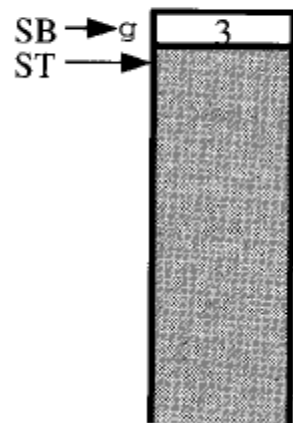
```
let var g: Integer;

  func F (m: Integer, n: Integer) : Integer ~
    m * n:

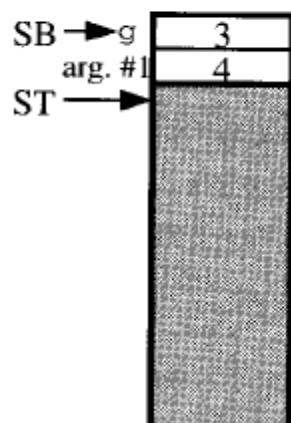
  proc W (i: Integer) ~
    let const s ~ i * i
    in
      begin
        putint(F(i, s));
        putint(F(s, s))
      end

in
  begin
    getint(var g);
    W(g+1)
  end
```

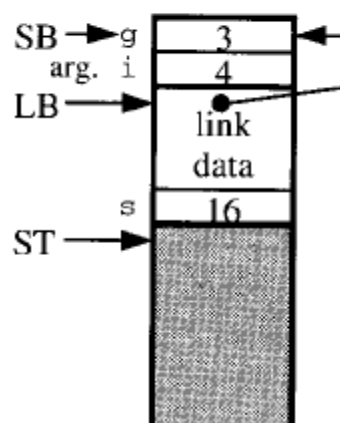
(1) Just after reading
g:



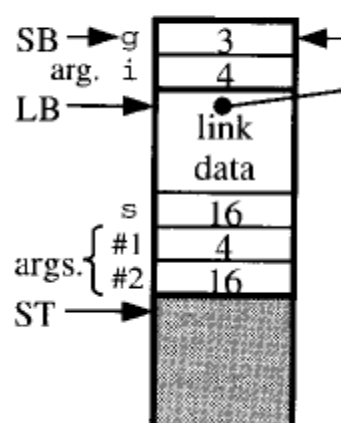
(2) Just before call to
W:



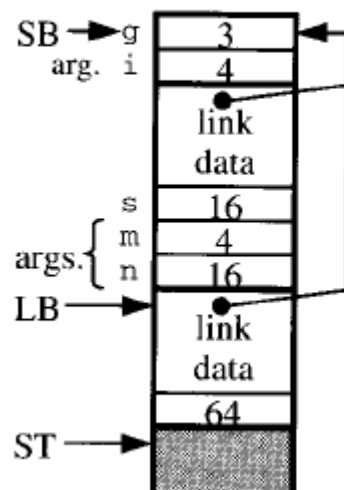
(3) Just after
computing s:



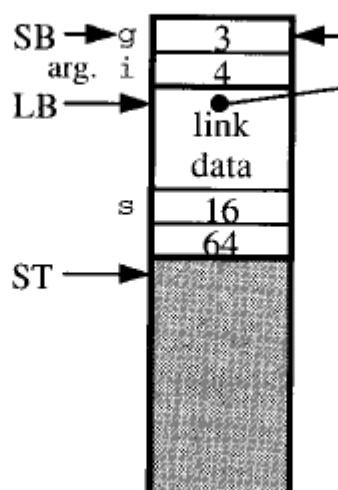
(4) Just before call to
F:



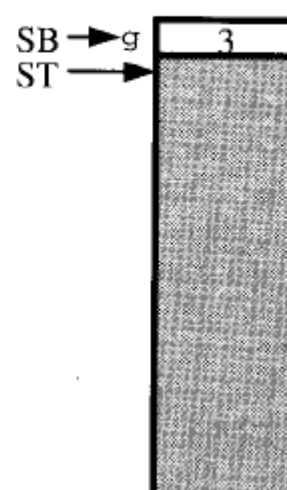
(5) Just before return
from F:



(6) Just after return
from F:



(7) Just after return
from W:



PUSH	1	– expand globals to make space for g
LOADA	0 [SB]	– push the <i>address</i> of g
CALL	<i>getint</i>	– read an integer into g
LOAD	0 [SB]	– push the <i>value</i> of g
CALL	<i>succ</i>	– 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>i</i>
LOAD	-1 [LB]	- push the value of <i>i</i>
CALL	<i>mult</i>	- multiply; the result will be the value of <i>s</i>
LOAD	-1 [LB]	- push the value of <i>i</i>
LOAD	3 [LB]	- push the value of <i>s</i>
CALL (SB)	<i>F</i>	- call <i>F</i> (using SB as static link)
CALL	<i>putint</i>	- write the value of <i>F(i, s)</i>
LOAD	3 [LB]	- push the value of <i>s</i>
LOAD	3 [LB]	- push the value of <i>s</i>
CALL (SB)	<i>F</i>	- call <i>F</i> (using SB as static link)
CALL	<i>putint</i>	- write the value of <i>F(s, s)</i>
RETURN (0)	1	- return, replacing the 1-word argument by a 0-word 'result'

<i>F</i> :	LOAD	-2 [LB]	- push the value of m
	LOAD	-1 [LB]	- push the value of n
	CALL	<i>mult</i>	- 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

```

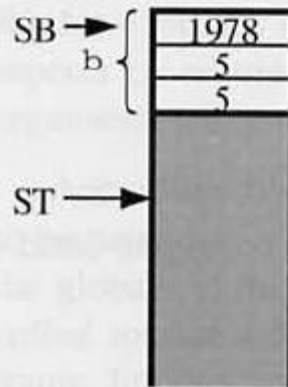
...

```

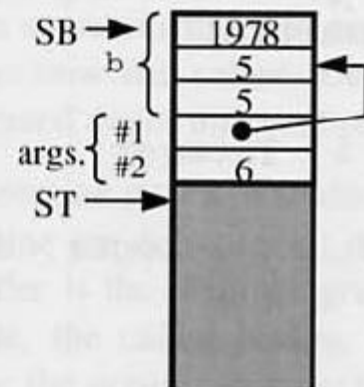
S: LOAD    -2[LB]    - push the argument address n
   LOADI    -        - push the value contained at that address
   LOAD    -1[LB]    - push the argument value i
   CALL    add        - add (giving the value of n+i)
   LOAD    -2[LB]    - push the argument address n
   STOREI    -        - store the value of n+i at that address
   RETURN(0) 2        - return, replacing the 2-word argument
                       pair by a 0-word 'result'

```

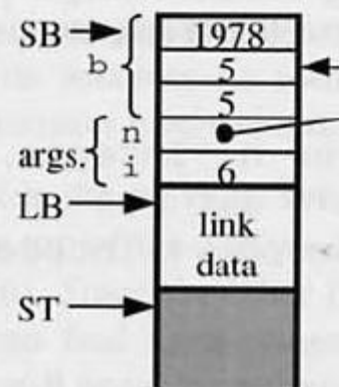
(1) Just after assignment to b:



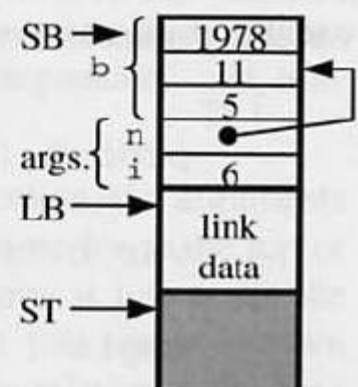
(2) Just before call to S:



(3) Just after entry to S:



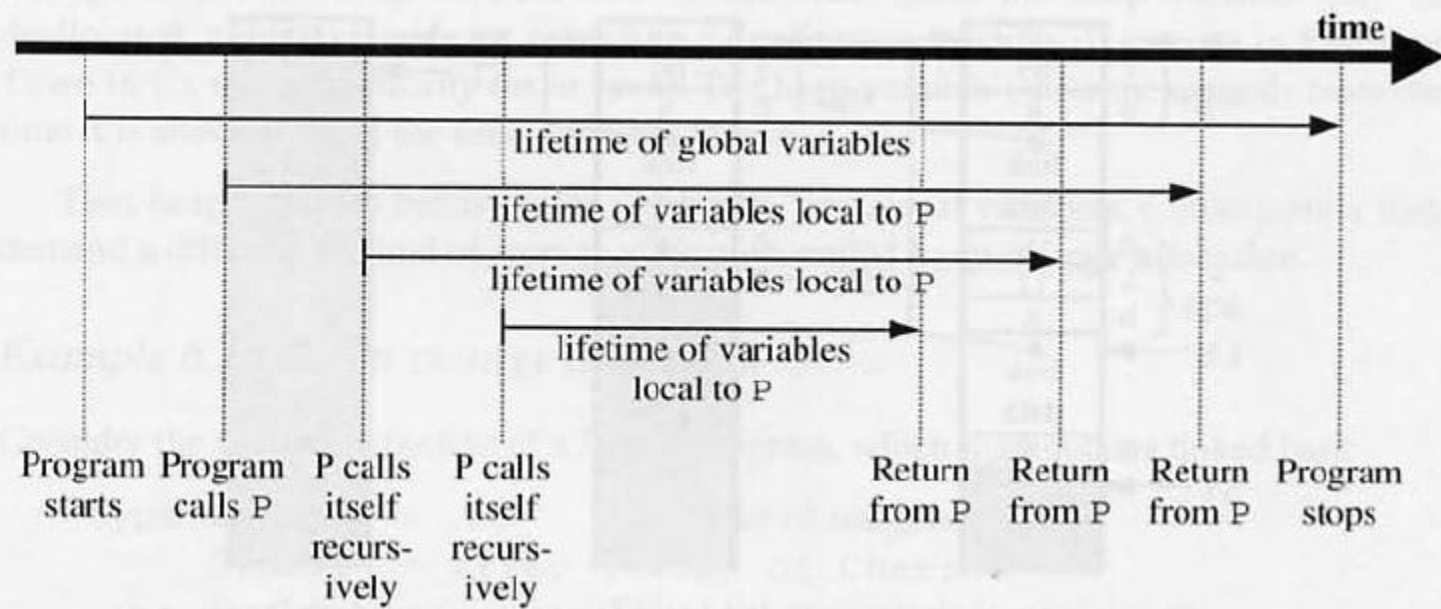
(4) Just before return from S:



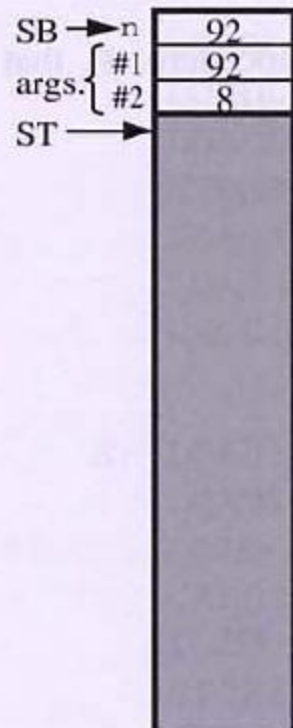
Recursão

- O programa do exemplo seguinte converte um número inteiro positivo decimal (base 10) para octal (base 8).
- Neste programa, onde se lê "`P (i // b, b)`" leia-se "`P (i / b, b)`".
- No exemplo, se 92_{10} é o número digitado, ele é convertido para 134_8 .

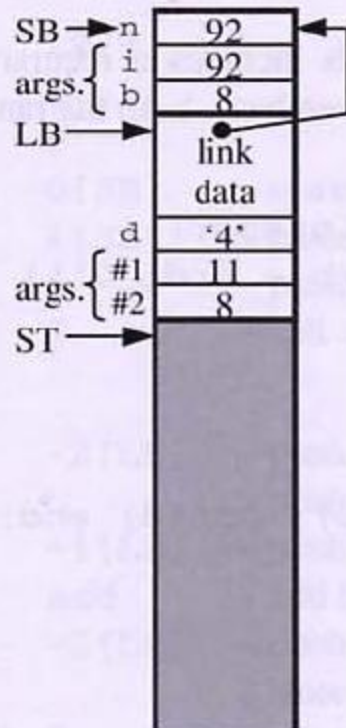
```
let
  proc P (i: Integer, b: Integer) ~
    let const d ~ chr(i//b + ord('0'))
    in
      if i < b then
        put(d)
      else
        begin P(i//b, b); put(d) end;
      var n: Integer
    in
      begin
        getint(var n); P(n, 8)
      end
```



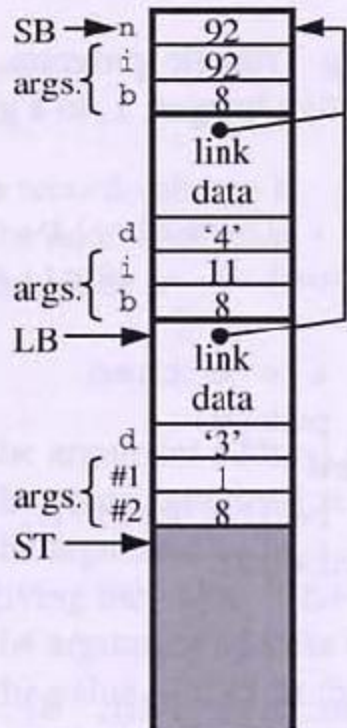
(1) Just before program calls P:



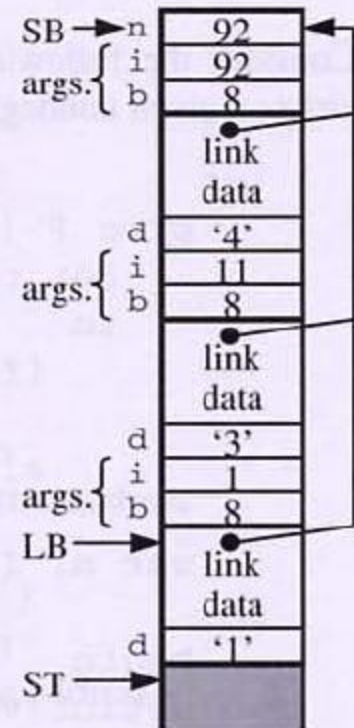
(2) Just before recursive call to P:



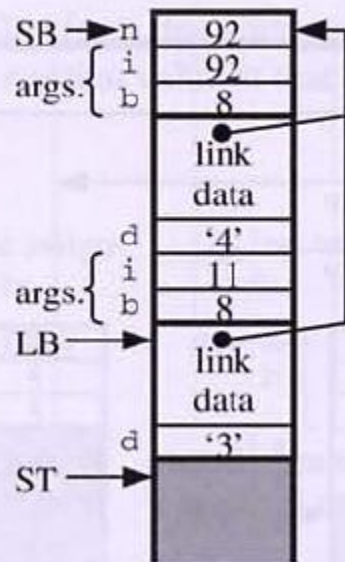
(3) Just before 2nd recursive call to P:



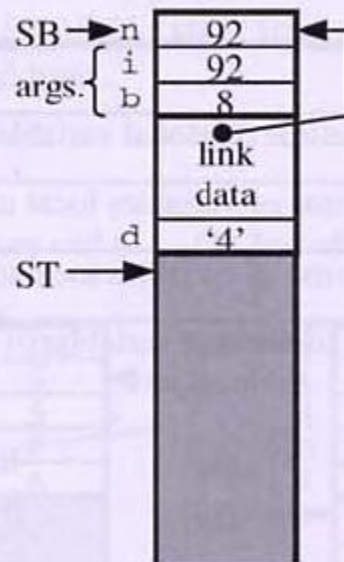
(4) Just after P computes d:



(5) After P writes '1' and returns:



(6) After P writes '3' and returns:



(7) After P writes '4' and returns:

