

Compiler Construction

Lecture 17: Code Generation II (The Compiler)

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)

RWTH Aachen University

noll@cs.rwth-aachen.de

<http://www-i2.informatik.rwth-aachen.de/i2/cc12/>

Summer Semester 2012

Online Registration for Seminars and Practical Courses (Praktika) in Winter Term 2012/13

Who?

- Students of:
- Master Courses
 - Bachelor Informatik (~~Pro~~Seminar!)

Where?

web-info8.informatik.rwth-aachen.de/apse

When?

18.06.2012 - 01.07.2012

- 1 Repetition: Intermediate Code
- 2 Semantics of Procedure and Transfer Instructions
- 3 The Symbol Table
- 4 Translation of Programs
- 5 Translation of Blocks
- 6 Translation of Declarations
- 7 Translation of Commands
- 8 Translation of Expressions
- 9 A Translation Example
- 10 Correctness of the Translation
- 11 Outlook

Definition (Syntax of EPL)

The **syntax of EPL** is defined as follows:

$\mathbb{Z} :$ z (* z is an integer *)

$Ide :$ I (* I is an identifier *)

$AExp :$ $A ::= z \mid I \mid A_1 + A_2 \mid \dots$

$BExp :$ $B ::= A_1 < A_2 \mid \text{not } B \mid B_1 \text{ and } B_2 \mid B_1 \text{ or } B_2$

$Cmd :$ $C ::= I := A \mid C_1 ; C_2 \mid \text{if } B \text{ then } C_1 \text{ else } C_2 \mid$
 $\text{while } B \text{ do } C \mid I()$

$Dcl :$ $D ::= D_C D_V D_P$

$D_C ::= \varepsilon \mid \text{const } l_1 := z_1, \dots, l_n := z_n;$

$D_V ::= \varepsilon \mid \text{var } l_1, \dots, l_n;$

$D_P ::= \varepsilon \mid \text{proc } l_1; K_1; \dots; \text{proc } l_n; K_n;$

$Blk :$ $K ::= D C$

$Pgm :$ $P ::= \text{in/out } l_1, \dots, l_n; K.$

Definition (Abstract machine for EPL)

The **abstract machine for EPL (AM)** is defined by the **state space**

$$S := PC \times DS \times PS$$

with

- the **program counter** $PC := \mathbb{N}$,
- the **data stack** $DS := \mathbb{Z}^*$ (top of stack to the right), and
- the **procedure stack** (or: **runtime stack**) $PS := \mathbb{Z}^*$ (top of stack to the left).

Thus a state $s = (l, d, p) \in S$ is given by

- a program label $l \in PC$,
- a data stack $d = d.r : \dots : d.1 \in DS$, and
- a procedure stack $p = p.1 : \dots : p.t \in PS$.

Structure of Procedure Stack I

The semantics of procedure and transfer instructions requires a particular structure of the procedure stack $p \in PS$: it must be composed of **frames** (or: **activation records**) of the form

$$sl : dl : ra : v_1 : \dots : v_k$$

where

static link sl : points to frame of surrounding declaration environment
 $\implies$ used to access non-local variables

dynamic link dl : points to previous frame (i.e., of calling procedure)
 $\implies$ used to remove topmost frame after termination of procedure call

return address ra : program label after termination of procedure call
 $\implies$ used to continue program execution after termination of procedure call

local variables v_i : values of locally declared variables

Structure of Procedure Stack IV

Observation:

- The usage of a variable in a procedure body refers to its **innermost declaration**.
- If the level difference between the usage and the declaration is **dif**, then a **chain of dif static links** has to be followed to access the corresponding frame.

Structure of Procedure Stack IV

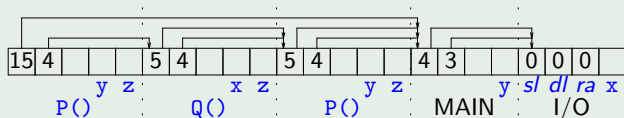
Observation:

- The usage of a variable in a procedure body refers to its **innermost declaration**.
- If the level difference between the usage and the declaration is **dif**, then a **chain of dif static links** has to be followed to access the corresponding frame.

Example (cf. Example 16.9)

```
in/out x;  
const c = 10;  
var y;  
proc P;  
  var y, z;  
  proc Q;  
    var x, z;  
    [... P() ...]  
  [... x ... y ... Q() ...]  
  proc R;  
    [... P() ...]  
  [... P() ...].
```

Procedure stack after second call of P:



Structure of Procedure Stack IV

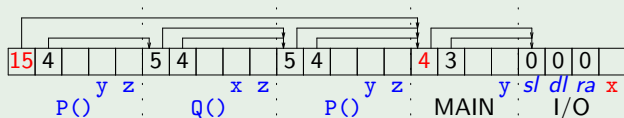
Observation:

- The usage of a variable in a procedure body refers to its **innermost declaration**.
- If the level difference between the usage and the declaration is **dif**, then a **chain of dif static links** has to be followed to access the corresponding frame.

Example (cf. Example 16.9)

```
in/out x;  
const c = 10;  
var y;  
proc P;  
  var y, z;  
  proc Q;  
    var x, z;  
    [... P() ...]  
  [... x ... y ... Q() ...]  
  proc R;  
    [... P() ...]  
  [... P() ...].
```

Procedure stack after second call of P:



P uses x $\Rightarrow$ dif = 2

Structure of Procedure Stack IV

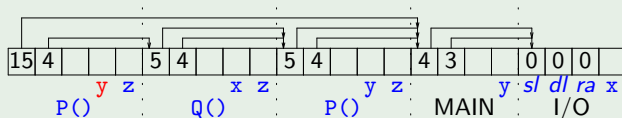
Observation:

- The usage of a variable in a procedure body refers to its **innermost declaration**.
- If the level difference between the usage and the declaration is **dif**, then a **chain of dif static links** has to be followed to access the corresponding frame.

Example (cf. Example 16.9)

```
in/out x;  
const c = 10;  
var y;  
proc P;  
  var y, z;  
  proc Q;  
    var x, z;  
    [... P() ...]  
  [... x ... y ... Q() ...]  
  proc R;  
    [... P() ...]  
  [... P() ...].
```

Procedure stack after second call of P:



P uses y $\Rightarrow$ dif = 0

- 1 Repetition: Intermediate Code
- 2 Semantics of Procedure and Transfer Instructions
- 3 The Symbol Table
- 4 Translation of Programs
- 5 Translation of Blocks
- 6 Translation of Declarations
- 7 Translation of Commands
- 8 Translation of Expressions
- 9 A Translation Example
- 10 Correctness of the Translation
- 11 Outlook

Semantics of Procedure Instructions

- **CALL**(*ca*, *dif*, *loc*) with

- code address *ca* $\in PC$
- level difference *dif* $\in \mathbb{N}$
- number of local variables *loc* $\in \mathbb{N}$

creates the new frame and jumps to the given address
(= starting address of procedure)

- **RET** removes the topmost frame and returns to the calling site

Semantics of Procedure Instructions

- **CALL**(*ca*, *dif*, *loc*) with

- **code address** *ca* $\in PC$
- **level difference** *dif* $\in \mathbb{N}$
- **number of local variables** *loc* $\in \mathbb{N}$

creates the new frame and **jumps** to the given address
(= starting address of procedure)

- **RET** **removes** the topmost frame and returns to the calling site

Definition 17.1 (Semantics of procedure instructions)

The **semantics of a procedure instruction** O , $\llbracket O \rrbracket : S \dashrightarrow S$, is defined as follows:

$$\begin{aligned} & \llbracket \text{CALL}(ca, dif, loc) \rrbracket(l, d, p) \\ & := (ca, d, \underbrace{(\text{base}(p, dif) + loc + 2)}_{sl} : \underbrace{(loc + 2)}_{dl} : \underbrace{(l + 1)}_{ra} : \underbrace{0 : \dots : 0}_{loc \text{ times}} : p) \end{aligned}$$

$$\begin{aligned} & \llbracket \text{RET} \rrbracket(l, d, p.1 : \dots : p.t) \\ & := (\underbrace{p.3}_{ra}, d, p.(\underbrace{p.2}_{dl} + 2) : \dots : p.t) \quad \text{if } t \geq p.2 + 2 \end{aligned}$$

Semantics of Transfer Instructions

- $\text{LOAD}(dif, off)$ and $\text{STORE}(dif, off)$ with

- level difference $dif \in \mathbb{N}$
- variable offset $off \in \mathbb{N}$

respectively load and store variable values between data and procedure stack, following a chain of dif static links

- $\text{LIT}(z)$ loads the literal constant $z \in \mathbb{Z}$

Semantics of Transfer Instructions

- $\text{LOAD}(dif, off)$ and $\text{STORE}(dif, off)$ with

- level difference $dif \in \mathbb{N}$
- variable offset $off \in \mathbb{N}$

respectively load and store variable values between data and procedure stack, following a chain of dif static links

- $\text{LIT}(z)$ loads the literal constant $z \in \mathbb{Z}$

Definition 17.2 (Semantics of transfer instructions)

The semantics of a transfer instruction O , $\llbracket O \rrbracket : S \dashrightarrow S$, is defined as follows:

$$\begin{aligned}\llbracket \text{LOAD}(dif, off) \rrbracket(l, d, p) &:= (l + 1, d : p.(\text{base}(p, dif) + off + 2), p) \\ \llbracket \text{STORE}(dif, off) \rrbracket(l, d : z, p) &:= (l + 1, d, p[\text{base}(p, dif) + off + 2 \mapsto z]) \\ \llbracket \text{LIT}(z) \rrbracket(l, d, p) &:= (l + 1, d : z, p)\end{aligned}$$

Definition 17.3 (Semantics of AM programs)

An **AM program** is a sequence of $k \geq 1$ labeled AM instructions:

$$P = 1 : O_1; \dots; k : O_k$$

The set of all AM programs is denoted by **AM**.

Definition 17.3 (Semantics of AM programs)

An **AM program** is a sequence of $k \geq 1$ labeled AM instructions:

$$P = 1 : O_1; \dots; k : O_k$$

The set of all AM programs is denoted by **AM**.

The **semantics of AM programs** is determined by

$$\llbracket \cdot \rrbracket : AM \times S \dashrightarrow S$$

with

$$\llbracket P \rrbracket(l, d, p) := \begin{cases} \llbracket P \rrbracket(\llbracket O_l \rrbracket(l, d, p)) & \text{if } l \in [k] \\ (l, d, p) & \text{otherwise} \end{cases}$$

- 1 Repetition: Intermediate Code
- 2 Semantics of Procedure and Transfer Instructions
- 3 The Symbol Table**
- 4 Translation of Programs
- 5 Translation of Blocks
- 6 Translation of Declarations
- 7 Translation of Commands
- 8 Translation of Expressions
- 9 A Translation Example
- 10 Correctness of the Translation
- 11 Outlook

Structure of Symbol Table

Goal: define translation mapping $\text{trans} : Pgm \dashrightarrow AM$

Structure of Symbol Table

Goal: define **translation mapping** $\text{trans} : \text{Pgm} \dashrightarrow \text{AM}$

The translation employs a **symbol table**:

$$\begin{aligned} \text{Tab} := \{ \text{st} \mid \text{st} : \text{Ide} \dashrightarrow & (\{\text{const}\} \times \mathbb{Z}) \\ & \cup (\{\text{var}\} \times \text{Lev} \times \text{Off}) \\ & \cup (\{\text{proc}\} \times \text{PC} \times \text{Lev} \times \text{Size}) \} \end{aligned}$$

whose entries are created by declarations:

- constant declarations: (const, z)
 - **value** $z \in \mathbb{Z}$
- variable declarations: $(\text{var}, \text{lev}, \text{off})$
 - **declaration level** $\text{lev} \in \text{Lev} := \mathbb{N}$ ($0 \cong \text{I/O}$, $1 \cong \text{MAIN}$, ...)
 - **offset** $\text{off} \in \text{Off} := \mathbb{N}$
 - offset and difference between usage and declaration level determine procedure stack entry
- procedure declarations: $(\text{proc}, \text{ca}, \text{lev}, \text{loc})$
 - **code address** $\text{ca} \in \text{PC}$
 - **declaration level** $\text{lev} \in \text{Lev}$
 - **number of local variables** $\text{loc} \in \text{Size} := \mathbb{N}$

Maintaining the Symbol Table

The symbol table is maintained by the function `update( $D$ ,  $st$ ,  $l$ )` which specifies the update of symbol table st according to declaration D (with respect to current level l):

Maintaining the Symbol Table

The symbol table is maintained by the function $\text{update}(D, \text{st}, l)$ which specifies the update of symbol table st according to declaration D (with respect to current level l):

Definition 17.4 (update function)

is defined by $\text{update} : Dcl \times Tab \times Lev \dashrightarrow Tab$

$$\begin{aligned} & \text{update}(D_C \ D_V \ D_P, \text{st}, l) \\ & \quad := \text{update}(D_P, \text{update}(D_V, \text{update}(D_C, \text{st}, l), l), l) \\ & \quad \quad \text{if all identifiers in } D_C \ D_V \ D_P \text{ different} \\ & \text{update}(\varepsilon, \text{st}, l) \\ & \quad := \text{st} \\ & \text{update}(\text{const } l_1 := z_1, \dots, l_n := z_n; \text{st}, l) \\ & \quad := \text{st}[l_1 \mapsto (\text{const}, z_1), \dots, l_n \mapsto (\text{const}, z_n)] \\ & \text{update}(\text{var } l_1, \dots, l_n; \text{st}, l) \\ & \quad := \text{st}[l_1 \mapsto (\text{var}, l, 1), \dots, l_n \mapsto (\text{var}, l, n)] \\ & \text{update}(\text{proc } l_1; K_1; \dots; \text{proc } l_n; K_n; \text{st}, l) \\ & \quad := \text{st}[l_1 \mapsto (\text{proc}, a_1, l, \text{size}(K_1)), \dots, l_n \mapsto (\text{proc}, a_n, l, \text{size}(K_n))] \\ & \quad \quad \text{with "fresh" addresses } a_1, \dots, a_n \\ & \quad \quad \text{where } \text{size}(D_C \text{ var } l_1, \dots, l_n; D_P C) := n \end{aligned}$$

Reminder: an EPL program $P = \text{in/out } l_1, \dots, l_n; K. \in Pgm$ has a **semantics** of type $\mathbb{Z}^n \dashrightarrow \mathbb{Z}^n$.

The Initial Symbol Table

Reminder: an EPL program $P = \text{in/out } l_1, \dots, l_n; K. \in Pgm$ has a **semantics** of type $\mathbb{Z}^n \dashrightarrow \mathbb{Z}^n$.

Given input values $(z_1, \dots, z_n) \in \mathbb{Z}^n$, we choose the **initial state**

$$s := (1, \varepsilon, \underbrace{0 : 0 : 0 : z_1 : \dots : z_n}_{\text{I/O frame}}) \in S = PC \times DS \times PS$$

The Initial Symbol Table

Reminder: an EPL program $P = \text{in/out } l_1, \dots, l_n; K. \in \text{Pgm}$ has a **semantics** of type $\mathbb{Z}^n \dashrightarrow \mathbb{Z}^n$.

Given input values $(z_1, \dots, z_n) \in \mathbb{Z}^n$, we choose the **initial state**

$$s := (1, \varepsilon, \underbrace{0 : 0 : 0 : z_1 : \dots : z_n}_{\text{I/O frame}}) \in S = PC \times DS \times PS$$

Thus the corresponding **initial symbol table** has n entries:

$$\text{st}_{I/O}(l_j) := (\text{var}, 0, j) \quad \text{for every } j \in [n]$$

- 1 Repetition: Intermediate Code
- 2 Semantics of Procedure and Transfer Instructions
- 3 The Symbol Table
- 4 Translation of Programs**
- 5 Translation of Blocks
- 6 Translation of Declarations
- 7 Translation of Commands
- 8 Translation of Expressions
- 9 A Translation Example
- 10 Correctness of the Translation
- 11 Outlook

Translation of `in/out  $l_1, \dots, l_n; D$   $C$  .:`

- 1 Create MAIN frame for executing C
- 2 Stop program execution after return

Translation of $\text{in/out } l_1, \dots, l_n; D \ C.:$

- 1 Create MAIN frame for executing C
- 2 Stop program execution after return

Definition 17.5 (Translation of programs)

The mapping

$$\text{trans} : \text{Pgm} \dashrightarrow \text{AM}$$

is defined by

$$\begin{aligned} \text{trans}(\text{in/out } l_1, \dots, l_n; K.) &:= 1 : \text{CALL}(a, 0, \text{size}(K)); \\ &\quad 2 : \text{JMP}(0); \\ &\quad \text{kt}(K, \text{st}_{I/O}, a, 1) \end{aligned}$$

- 1 Repetition: Intermediate Code
- 2 Semantics of Procedure and Transfer Instructions
- 3 The Symbol Table
- 4 Translation of Programs
- 5 Translation of Blocks**
- 6 Translation of Declarations
- 7 Translation of Commands
- 8 Translation of Expressions
- 9 A Translation Example
- 10 Correctness of the Translation
- 11 Outlook

Translation of Blocks

Translation of D C :

- 1 Update symbol table according to D
- 2 Create code for procedures declared in D
(using the updated symbol table – recursion!)
- 3 Create code for C (using the updated symbol table)

Translation of Blocks

Translation of $D \ C$:

- 1 Update symbol table according to D
- 2 Create code for procedures declared in D
(using the updated symbol table – recursion!)
- 3 Create code for C (using the updated symbol table)

Definition 17.6 (Translation of blocks)

The mapping

$$kt : Blk \times Tab \times PC \times Lev \dashrightarrow AM$$

(“block translation”) is defined by

$$\begin{aligned} kt(D \ C, st, a, l) &:= dt(D, update(D, st, l), l) \\ &\quad ct(C, update(D, st, l), a, l) \\ &\quad a' : RET; \end{aligned}$$

- 1 Repetition: Intermediate Code
- 2 Semantics of Procedure and Transfer Instructions
- 3 The Symbol Table
- 4 Translation of Programs
- 5 Translation of Blocks
- 6 Translation of Declarations**
- 7 Translation of Commands
- 8 Translation of Expressions
- 9 A Translation Example
- 10 Correctness of the Translation
- 11 Outlook

Translation of Declarations

Translation of D : generate code for the procedures declared in D

Translation of Declarations

Translation of D : generate code for the procedures declared in D

Definition 17.7 (Translation of declarations)

The mapping

$$dt : Dcl \times Tab \times Lev \dashrightarrow AM$$

(“declaration translation”) is defined by

$$\begin{aligned} dt(D_C \ D_V \ D_P, st, l) &:= dt(D_P, st, l) \\ dt(\varepsilon, st, l) &:= \varepsilon \\ dt(\text{proc } l_1; K_1; \dots; \text{proc } l_n; K_n; , st, l) &:= kt(K_1, st, a_1, l + 1) \\ &\quad \vdots \\ &\quad kt(K_n, st, a_n, l + 1) \\ &\quad \text{where } st(l_j) = (\text{proc}, a_j, \dots, \dots) \text{ for every } j \in [n] \end{aligned}$$

- 1 Repetition: Intermediate Code
- 2 Semantics of Procedure and Transfer Instructions
- 3 The Symbol Table
- 4 Translation of Programs
- 5 Translation of Blocks
- 6 Translation of Declarations
- 7 Translation of Commands**
- 8 Translation of Expressions
- 9 A Translation Example
- 10 Correctness of the Translation
- 11 Outlook

Definition 17.8 (Translation of commands)

The mapping

$$ct : Cmd \times Tab \times PC \times Lev \dashrightarrow AM$$

(“command translation”) is defined by

$$\begin{aligned} ct(l := A, st, a, l) &:= at(A, st, a, l) \\ &\quad a' : STORE(l - lev, off); \\ &\quad if\ st(l) = (var, lev, off) \\ ct(l(), st, a, l) &:= a : CALL(ca, l - lev, loc); \\ &\quad if\ st(l) = (proc, ca, lev, loc) \\ ct(C_1; C_2, st, a, l) &:= ct(C_1, st, a, l) \\ &\quad ct(C_2, st, a', l) \\ ct(if\ B\ then\ C_1\ else\ C_2, st, a, l) &:= bt(B, st, a, l) \\ &\quad a' : JFALSE(a''); \\ &\quad ct(C_1, st, a' + 1, l) \\ &\quad a'' - 1 : JMP(a'''); \\ &\quad ct(C_2, st, a'', l) \\ &\quad a''' : \\ ct(while\ B\ do\ C, st, a, l) &:= bt(B, st, a, l) \\ &\quad a' : JFALSE(a'' + 1); \\ &\quad ct(C, st, a' + 1, l) \\ &\quad a'' : JMP(a); \end{aligned}$$

- 1 Repetition: Intermediate Code
- 2 Semantics of Procedure and Transfer Instructions
- 3 The Symbol Table
- 4 Translation of Programs
- 5 Translation of Blocks
- 6 Translation of Declarations
- 7 Translation of Commands
- 8 Translation of Expressions**
- 9 A Translation Example
- 10 Correctness of the Translation
- 11 Outlook

Definition 17.9 (Translation of Boolean expressions)

The mapping

$$bt : BExp \times Tab \times PC \times Lev \dashrightarrow AM$$

(“Boolean expression translation”) is defined by

$$\begin{aligned} bt(A_1 < A_2, st, a, l) &:= at(A_1, st, a, l) \\ &\quad at(A_2, st, a', l) \\ &\quad a'' : LT; \end{aligned}$$

$$\begin{aligned} bt(\text{not } B, st, a, l) &:= bt(B, st, a, l) \\ &\quad a' : NOT; \end{aligned}$$

$$\begin{aligned} bt(B_1 \text{ and } B_2, st, a, l) &:= bt(B_1, st, a, l) \\ &\quad bt(B_2, st, a', l) \\ &\quad a'' : AND; \end{aligned}$$

$$\begin{aligned} bt(B_1 \text{ or } B_2, st, a, l) &:= bt(B_1, st, a, l) \\ &\quad bt(B_2, st, a', l) \\ &\quad a'' : OR; \end{aligned}$$

Definition 17.10 (Translation of arithmetic expressions)

The mapping

$$at : AExp \times Tab \times PC \times Lev \dashrightarrow AM$$

(“arithmetic expression translation”) is defined by

$$at(z, st, a, l) := a : LIT(z) ;$$

$$at(l, st, a, l) := \begin{cases} a : LIT(z) ; & \text{if } st(l) = (const, z) \\ a : LOAD(l - lev, off) ; & \text{if } st(l) = (var, lev, off) \end{cases}$$

$$at(A_1 + A_2, st, a, l) := \begin{aligned} &at(A_1, st, a, l) \\ &at(A_2, st, a', l) \\ &a'' : ADD ; \end{aligned}$$

- 1 Repetition: Intermediate Code
- 2 Semantics of Procedure and Transfer Instructions
- 3 The Symbol Table
- 4 Translation of Programs
- 5 Translation of Blocks
- 6 Translation of Declarations
- 7 Translation of Commands
- 8 Translation of Expressions
- 9 A Translation Example**
- 10 Correctness of the Translation
- 11 Outlook

Example 17.11 (Factorial function; cf. Example 16.3)

Source code:

```
in/out x;  
var y;  
proc F;  
  if x > 1 then  
    y := y * x;  
    x := x - 1;  
    F()  
  y := 1;  
  F();  
  x := y.
```

Intermediate code:

Example: Factorial Function I

Example 17.11 (Factorial function; cf. Example 16.3)

Source code:

```
in/out x;  
  var y;  
  proc F;  
    if x > 1 then  
      y := y * x;  
      x := x - 1;  
      F()  
    y := 1;  
    F();  
    x := y.  
trans(in/out  $l_1, \dots, l_n; K.$ ) :=  
  1 : CALL(a, 0, size(K));  
  2 : JMP(0);  
  kt(K, stI/O, a, 1)  
stI/O = [x  $\mapsto$  (var, 0, 1)]
```

Intermediate code:

```
trans(in/out x; K.)
```

Example: Factorial Function I

Example 17.11 (Factorial function; cf. Example 16.3)

Source code:

```
in/out x;  
var y;  
proc F;  
  if x > 1 then  
    y := y * x;  
    x := x - 1;  
    F()  
  y := 1;  
  F();  
  x := y.  
kt( $D \ C$ , st, a, l) :=  
  dt( $D$ , update( $D$ , st, l), l)  
  ct( $C$ , update( $D$ , st, l), a, l)  
  a' : RET;  
stl/o = [x  $\mapsto$  (var, 0, 1)]
```

Intermediate code:

```
1 : CALL(a0, 0, 1);  
2 : JMP(0);  
   kt( $K$ , stl/o, a0, 1)
```

Example: Factorial Function I

Example 17.11 (Factorial function; cf. Example 16.3)

Source code:

```
in/out x;  
var y;  
proc F;  
  if x > 1 then  
    y := y * x;  
    x := x - 1;  
    F()  
  y := 1;  
  F();  
  x := y.
```

$\text{update}(\text{var } l_1, \dots, l_n; \text{st}, l) :=$

$\text{st}[l_1 \mapsto (\text{var}, l, 1), \dots, l_n \mapsto (\text{var}, l, n)]$

$\text{update}(\text{proc } l_1; K_1; \dots; \text{proc } l_n; K_n; \text{st}, l) :=$

$\text{st}[l_1 \mapsto (\text{proc}, a_1, l, \text{size}(K_1)), \dots, l_n \mapsto (\text{proc}, a_n, l, \text{size}(K_n))]$

$\text{st}' = [x \mapsto (\text{var}, 0, 1),$

$y \mapsto (\text{var}, 1, 1),$

$F \mapsto (\text{proc}, a_1, 1, 0)]$

Intermediate code:

```
1 : CALL(a0, 0, 1);  
2 : JMP(0);  
   dt(D, update(D, stI/O, 1), 1)  
   ct(C, update(D, stI/O, 1), a0, 1)  
a2 : RET;
```

Example: Factorial Function I

Example 17.11 (Factorial function; cf. Example 16.3)

Source code:

```
in/out x;  
var y;  
proc F;  
  if x > 1 then  
    y := y * x;  
    x := x - 1;  
    F()  
  y := 1;  
  F();  
x := y.
```

$dt(\text{proc } h_1; K_1; \dots; \text{proc } h_n; K_n; , st, l) :=$
 $kt(K_1, st, a_1, l + 1)$

$\vdots$

$kt(K_n, st, a_n, l + 1)$

where $st(l_j) = (\text{proc}, a_j, \dots, \dots)$ for every $j \in [n]$

$st' = [x \mapsto (\text{var}, 0, 1),$
 $y \mapsto (\text{var}, 1, 1),$
 $F \mapsto (\text{proc}, a_1, 1, 0)]$

Intermediate code:

```
1 : CALL(a0, 0, 1);  
2 : JMP(0);  
    dt(D, st', 1)  
    ct(C, st', a0, 1)  
a2 : RET;
```

Example: Factorial Function I

Example 17.11 (Factorial function; cf. Example 16.3)

Source code:

```
in/out x;  
var y;  
proc F;  
  if x > 1 then  
    y := y * x;  
    x := x - 1;  
    F()  
  y := 1;  
  F();  
  x := y.
```

```
kt( $D$   $C$ , st, a, l) :=  
  dt( $D$ , update( $D$ , st, l), l)  
  ct( $C$ , update( $D$ , st, l), a, l)  
  a' : RET;
```

```
st' = [x  $\mapsto$  (var, 0, 1),  
       y  $\mapsto$  (var, 1, 1),  
       F  $\mapsto$  (proc, a1, 1, 0)]
```

Intermediate code:

```
1 : CALL(a0, 0, 1);  
2 : JMP(0);  
   kt( $K_F$ , st', a1, 2)  
   ct( $C$ , st', a0, 1)  
a2 : RET;
```

Example: Factorial Function I

Example 17.11 (Factorial function; cf. Example 16.3)

Source code:

```
in/out x;  
var y;  
proc F;  
  if x > 1 then  
    y := y * x;  
    x := x - 1;  
    F()  
  y := 1;  
  F();  
x := y.
```

```
ct(if B then C1 else C2, st, a, l) :=  
  bt(B, st, a, l)  
  a' : JFALSE(a'');  
  ct(C1, st, a' + 1, l)  
  a'' - 1 : JMP(a''');  
  ct(C2, st, a'', l)  
  a''' :
```

```
st' = [x ↦ (var, 0, 1),  
       y ↦ (var, 1, 1),  
       F ↦ (proc, a1, 1, 0)]
```

Intermediate code:

```
1 : CALL(a0, 0, 1);  
2 : JMP(0);  
   ct(CF, st', a1, 2)  
a3 : RET;  
   ct(C, st', a0, 1)  
a2 : RET;
```

Example: Factorial Function I

Example 17.11 (Factorial function; cf. Example 16.3)

Source code:

```
in/out x;  
var y;  
proc F;  
  if x > 1 then  
    y := y * x;  
    x := x - 1;  
    F()  
  y := 1;  
  F();  
  x := y.
```

$bt(A_1 > A_2, st, a, l) := at(A_1, st, a, l)$
 $at(A_2, st, a', l)$
 $a'' : GT;$

$st' = [x \mapsto (var, 0, 1),$
 $y \mapsto (var, 1, 1),$
 $F \mapsto (proc, a_1, 1, 0)]$

Intermediate code:

```
1 : CALL(a0, 0, 1);  
2 : JMP(0);  
   bt(x > 1, st', a1, 2)  
a4 : JFALSE(a3);  
   ct(C1, st', a4 + 1, 2)  
a3 : RET;  
   ct(C, st', a0, 1)  
a2 : RET;
```

Example: Factorial Function I

Example 17.11 (Factorial function; cf. Example 16.3)

Source code:

```
in/out x;  
var y;  
proc F;  
  if x > 1 then  
    y := y * x;  
    x := x - 1;  
    F()  
  y := 1;  
  F();  
  x := y.
```

$at(l, st, a, l) :=$

$$\begin{cases} a : LIT(z); & \text{if } st(l) = (const, z) \\ a : LOAD(l - lev, off); & \text{if } st(l) = (var, lev, off) \end{cases}$$
$$st' = [x \mapsto (var, 0, 1), \\ y \mapsto (var, 1, 1), \\ F \mapsto (proc, a_1, 1, 0)]$$

Intermediate code:

```
1 : CALL(a0, 0, 1);  
2 : JMP(0);  
   at(x, st', a1, 2)  
   at(1, st', a', 2)  
   GT;  
a4 : JFALSE(a3);  
      ct(C1, st', a4 + 1, 2)  
a3 : RET;  
      ct(C, st', a0, 1)  
a2 : RET;
```

Example: Factorial Function I

Example 17.11 (Factorial function; cf. Example 16.3)

Source code:

```
in/out x;  
var y;  
proc F;  
  if x > 1 then  
    y := y * x;  
    x := x - 1;  
    F()  
  y := 1;  
  F();  
  x := y.  
  
at(z, st, a, l) := a : LIT(z);  
  
st' = [x ↦ (var, 0, 1),  
       y ↦ (var, 1, 1),  
       F ↦ (proc, a1, 1, 0)]
```

Intermediate code:

```
1 : CALL(a0, 0, 1);  
2 : JMP(0);  
a1 : LOAD(2, 1);  
      at(1, st', a', 2)  
      GT;  
a4 : JFALSE(a3);  
      ct(C1, st', a4 + 1, 2)  
a3 : RET;  
      ct(C, st', a0, 1)  
a2 : RET;
```

Example: Factorial Function I

Example 17.11 (Factorial function; cf. Example 16.3)

Source code:

```
in/out x;  
var y;  
proc F;  
  if x > 1 then  
    y := y * x;  
    x := x - 1;  
    F()  
  y := 1;  
  F();  
  x := y.  
  
ct(l := A, st, a, l) :=  
  at(A, st, a, l)  
  a' : STORE(l - lev, off);  
  if st(l) = (var, lev, off)  
  
st' = [x ↦ (var, 0, 1),  
       y ↦ (var, 1, 1),  
       F ↦ (proc, a_1, 1, 0)]
```

Intermediate code:

```
1 : CALL(a_0, 0, 1);  
2 : JMP(0);  
a_1 : LOAD(2, 1);  
      LIT(1);  
      GT;  
a_4 : JFALSE(a_3);  
      ct(C_1, st', a_4 + 1, 2)  
a_3 : RET;  
      ct(C, st', a_0, 1)  
a_2 : RET;
```

Example: Factorial Function I

Example 17.11 (Factorial function; cf. Example 16.3)

Source code:

```
in/out x;  
var y;  
proc F;  
  if x > 1 then  
    y := y * x;  
    x := x - 1;  
    F()  
  y := 1;  
  F();  
  x := y.  
  
at(A1 + A2, st, a, l) := at(A1, st, a, l)  
                        at(A2, st, a', l)  
                        a'' : ADD;  
  
st' = [x ↦ (var, 0, 1),  
       y ↦ (var, 1, 1),  
       F ↦ (proc, a1, 1, 0)]
```

Intermediate code:

```
1 : CALL(a0, 0, 1);  
2 : JMP(0);  
a1 : LOAD(2, 1);  
      LIT(1);  
      GT;  
a4 : JFALSE(a3);  
      at(y * x, st', a4 + 1, 2)  
      STORE(1, 1);  
      at(x - 1, st', a', 2)  
      STORE(2, 1);  
      ct(F(), st', a'', 2)  
      a3 : RET;  
      ct(C, st', a0, 1)  
a2 : RET;
```

Example: Factorial Function I

Example 17.11 (Factorial function; cf. Example 16.3)

Source code:

```
in/out x;  
var y;  
proc F;  
  if x > 1 then  
    y := y * x;  
    x := x - 1;  
  F()  
y := 1;  
F();  
x := y.  
  
ct(I(), st, a, l) :=  
  a : CALL(ca, l - lev, loc);  
if st(l) = (proc, ca, lev, loc)  
st' = [x ↦ (var, 0, 1),  
       y ↦ (var, 1, 1),  
       F ↦ (proc, a1, 1, 0)]
```

Intermediate code:

```
1 : CALL(a0, 0, 1);  
2 : JMP(0);  
a1 : LOAD(2, 1);  
      LIT(1);  
      GT;  
a4 : JFALSE(a3);  
      LOAD(1, 1);  
      LOAD(2, 1);  
      MULT;  
      STORE(1, 1);  
      LOAD(2, 1);  
      LIT(1);  
      SUB;  
      STORE(2, 1);  
      ct(F(), st', a'', 2)  
a3 : RET;  
      ct(C, st', a0, 1)  
a2 : RET;
```

Example: Factorial Function I

Example 17.11 (Factorial function; cf. Example 16.3)

Source code:

```
in/out x;  
var y;  
proc F;  
  if x > 1 then  
    y := y * x;  
    x := x - 1;  
    F()  
  y := 1;  
  F();  
  x := y.
```

```
st' = [x ↦ (var, 0, 1),  
       y ↦ (var, 1, 1),  
       F ↦ (proc, a1, 1, 0)]
```

Intermediate code:

```
1 : CALL(a0, 0, 1);  
2 : JMP(0);  
a1 : LOAD(2, 1);  
      LIT(1);  
      GT;  
a4 : JFALSE(a3);  
      LOAD(1, 1);  
      LOAD(2, 1);  
      MULT;  
      STORE(1, 1);  
      LOAD(2, 1);  
      LIT(1);  
      SUB;  
      STORE(2, 1);  
      CALL(a1, 1, 0);  
a3 : RET;  
      ct(C, st', a0, 1)  
a2 : RET;
```

Example: Factorial Function I

Example 17.11 (Factorial function; cf. Example 16.3)

Source code:

```
in/out x;  
var y;  
proc F;  
  if x > 1 then  
    y := y * x;  
    x := x - 1;  
    F()  
  y := 1;  
  F();  
  x := y.
```

```
st' = [x ↦ (var, 0, 1),  
       y ↦ (var, 1, 1),  
       F ↦ (proc, a1, 1, 0)]
```

Intermediate code:

```
1 : CALL(a0, 0, 1);  
2 : JMP(0);  
a1 : LOAD(2, 1);  
      LIT(1);  
      GT;  
a4 : JFALSE(a3);  
      LOAD(1, 1);  
      LOAD(2, 1);  
      MULT;  
      STORE(1, 1);  
      LOAD(2, 1);  
      LIT(1);  
      SUB;  
      STORE(2, 1);  
      CALL(a1, 1, 0);  
a3 : RET;  
a0 : LIT(1);  
      STORE(0, 1);  
      CALL(a1, 0, 0);  
      LOAD(0, 1);  
      STORE(1, 1);  
a2 : RET;
```

Example: Factorial Function II

Example 17.11 (Factorial function; continued)

Code with symbolic addresses:

```
1 : CALL(a0,0,1);
2 : JMP(0);
a1 : LOAD(2,1);
    LIT(1);
    GT;
a4 : JFALSE(a3);
    LOAD(1,1);
    LOAD(2,1);
    MULT;
    STORE(1,1);
    LOAD(2,1);
    LIT(1);
    SUB;
    STORE(2,1);
    CALL(a1,1,0);
a3 : RET;
a0 : LIT(1);
    STORE(0,1);
    CALL(a1,0,0);
    LOAD(0,1);
    STORE(1,1);
a2 : RET;
```

Linearized ($a_0 = 17, a_1 = 3, a_2 = 22, a_3 = 16, a_4 = 6$):

```
1 : CALL(17,0,1);
2 : JMP(0);
3 : LOAD(2,1);
4 : LIT(1);
5 : GT;
6 : JFALSE(16);
7 : LOAD(1,1);
8 : LOAD(2,1);
9 : MULT;
10 : STORE(1,1);
11 : LOAD(2,1);
12 : LIT(1);
13 : SUB;
14 : STORE(2,1);
15 : CALL(3,1,0);
16 : RET;
17 : LIT(1);
18 : STORE(0,1);
19 : CALL(3,0,0);
20 : LOAD(0,1);
21 : STORE(1,1);
22 : RET;
```

Example: Factorial Function III

Example 17.11 (Factorial function; continued)

Computation for $x = 2$:

<i>PC</i>	<i>DS</i>	<i>PS</i>
1	ϵ	<u>0:0:0:2</u>

```
1: CALL(17,0,1);
2: JMP(0);
3: LOAD(2,1);
4: LIT(1);
5: GT;
6: JFALSE(16);
7: LOAD(1,1);
8: LOAD(2,1);
9: MULT;
10: STORE(1,1);
11: LOAD(2,1);
12: LIT(1);
13: SUB;
14: STORE(2,1);
15: CALL(3,1,0);
16: RET;
17: LIT(1);
18: STORE(0,1);
19: CALL(3,0,0);
20: LOAD(0,1);
21: STORE(1,1);
22: RET;
```

Example: Factorial Function III

Example 17.11 (Factorial function; continued)

Computation for $x = 2$:

	<i>PC</i>	<i>DS</i>	<i>PS</i>
1 : CALL(17,0,1);	<u>1</u>	<u>ϵ</u>	<u>0:0:0:2</u>
2 : JMP(0);	17	ϵ	<u>4:3:2:0:0:0:0:2</u>
3 : LOAD(2,1);			
4 : LIT(1);			
5 : GT;			
6 : JFALSE(16);			
7 : LOAD(1,1);			
8 : LOAD(2,1);			
9 : MULT;			
10 : STORE(1,1);			
11 : LOAD(2,1);			
12 : LIT(1);			
13 : SUB;			
14 : STORE(2,1);			
15 : CALL(3,1,0);			
16 : RET;			
17 : LIT(1);			
18 : STORE(0,1);			
19 : CALL(3,0,0);			
20 : LOAD(0,1);			
21 : STORE(1,1);			
22 : RET;			

Example: Factorial Function III

Example 17.11 (Factorial function; continued)

Computation for $x = 2$:

	<i>PC</i>	<i>DS</i>	<i>PS</i>
1 : CALL(17,0,1);	1	ϵ	0 : 0 : 0 : 2
2 : JMP(0);	17	ϵ	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
3 : LOAD(2,1);	18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
4 : LIT(1);			
5 : GT;			
6 : JFALSE(16);			
7 : LOAD(1,1);			
8 : LOAD(2,1);			
9 : MULT;			
10 : STORE(1,1);			
11 : LOAD(2,1);			
12 : LIT(1);			
13 : SUB;			
14 : STORE(2,1);			
15 : CALL(3,1,0);			
16 : RET;			
17 : LIT(1);			
18 : STORE(0,1);			
19 : CALL(3,0,0);			
20 : LOAD(0,1);			
21 : STORE(1,1);			
22 : RET;			

Example: Factorial Function III

Example 17.11 (Factorial function; continued)

Computation for $x = 2$:

	<i>PC</i>	<i>DS</i>	<i>PS</i>
1 : CALL(17,0,1);	1	ϵ	0 : 0 : 0 : 2
2 : JMP(0);	17	ϵ	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
3 : LOAD(2,1);	18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
4 : LIT(1);	19	ϵ	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5 : GT;			
6 : JFALSE(16);			
7 : LOAD(1,1);			
8 : LOAD(2,1);			
9 : MULT;			
10 : STORE(1,1);			
11 : LOAD(2,1);			
12 : LIT(1);			
13 : SUB;			
14 : STORE(2,1);			
15 : CALL(3,1,0);			
16 : RET;			
17 : LIT(1);			
18 : STORE(0,1);			
19 : CALL(3,0,0);			
20 : LOAD(0,1);			
21 : STORE(1,1);			
22 : RET;			

Example: Factorial Function III

Example 17.11 (Factorial function; continued)

Computation for $x = 2$:

	<u>PC</u>	<u>DS</u>	<u>PS</u>
1 : CALL(17,0,1);	1	ϵ	0 : 0 : 0 : 2
2 : JMP(0);	17	ϵ	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
3 : LOAD(2,1);	18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
4 : LIT(1);	19	ϵ	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5 : GT;	3	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6 : JFALSE(16);			
7 : LOAD(1,1);			
8 : LOAD(2,1);			
9 : MULT;			
10 : STORE(1,1);			
11 : LOAD(2,1);			
12 : LIT(1);			
13 : SUB;			
14 : STORE(2,1);			
15 : CALL(3,1,0);			
16 : RET;			
17 : LIT(1);			
18 : STORE(0,1);			
19 : CALL(3,0,0);			
20 : LOAD(0,1);			
21 : STORE(1,1);			
22 : RET;			

Example: Factorial Function III

Example 17.11 (Factorial function; continued)

Computation for $x = 2$:

	<i>PC</i>	<i>DS</i>	<i>PS</i>
1 : CALL(17,0,1);	1	ϵ	0 : 0 : 0 : 2
2 : JMP(0);	17	ϵ	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
3 : LOAD(2,1);	18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
4 : LIT(1);	19	ϵ	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5 : GT;	3	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6 : JFALSE(16);	4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7 : LOAD(1,1);			
8 : LOAD(2,1);			
9 : MULT;			
10 : STORE(1,1);			
11 : LOAD(2,1);			
12 : LIT(1);			
13 : SUB;			
14 : STORE(2,1);			
15 : CALL(3,1,0);			
16 : RET;			
17 : LIT(1);			
18 : STORE(0,1);			
19 : CALL(3,0,0);			
20 : LOAD(0,1);			
21 : STORE(1,1);			
22 : RET;			

Example: Factorial Function III

Example 17.11 (Factorial function; continued)

Computation for $x = 2$:

	<i>PC</i>	<i>DS</i>	<i>PS</i>
1 : CALL(17,0,1);	1	ϵ	0 : 0 : 0 : 2
2 : JMP(0);	17	ϵ	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
3 : LOAD(2,1);	18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
4 : LIT(1);	19	ϵ	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5 : GT;	3	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6 : JFALSE(16);	4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7 : LOAD(1,1);	5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8 : LOAD(2,1);			
9 : MULT;			
10 : STORE(1,1);			
11 : LOAD(2,1);			
12 : LIT(1);			
13 : SUB;			
14 : STORE(2,1);			
15 : CALL(3,1,0);			
16 : RET;			
17 : LIT(1);			
18 : STORE(0,1);			
19 : CALL(3,0,0);			
20 : LOAD(0,1);			
21 : STORE(1,1);			
22 : RET;			

Example: Factorial Function III

Example 17.11 (Factorial function; continued)

Computation for $x = 2$:

	<i>PC</i>	<i>DS</i>	<i>PS</i>
1 : CALL(17,0,1);	1	ϵ	0 : 0 : 0 : 2
2 : JMP(0);	17	ϵ	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
3 : LOAD(2,1);	18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
4 : LIT(1);	19	ϵ	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5 : GT;	3	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6 : JFALSE(16);	4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7 : LOAD(1,1);	5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8 : LOAD(2,1);	6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9 : MULT;			3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10 : STORE(1,1);			3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11 : LOAD(2,1);			3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
12 : LIT(1);			3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
13 : SUB;			3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
14 : STORE(2,1);			3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
15 : CALL(3,1,0);			3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
16 : RET;			3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
17 : LIT(1);			3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
18 : STORE(0,1);			3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
19 : CALL(3,0,0);			3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
20 : LOAD(0,1);			3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
21 : STORE(1,1);			3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
22 : RET;			3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2

Example: Factorial Function III

Example 17.11 (Factorial function; continued)

Computation for $x = 2$:

	<i>PC</i>	<i>DS</i>	<i>PS</i>
1 : CALL(17,0,1);	1	ϵ	0 : 0 : 0 : 2
2 : JMP(0);	17	ϵ	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
3 : LOAD(2,1);	18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
4 : LIT(1);	19	ϵ	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5 : GT;	3	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6 : JFALSE(16);	4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7 : LOAD(1,1);	5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8 : LOAD(2,1);	6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9 : MULT;	7	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10 : STORE(1,1);			
11 : LOAD(2,1);			
12 : LIT(1);			
13 : SUB;			
14 : STORE(2,1);			
15 : CALL(3,1,0);			
16 : RET;			
17 : LIT(1);			
18 : STORE(0,1);			
19 : CALL(3,0,0);			
20 : LOAD(0,1);			
21 : STORE(1,1);			
22 : RET;			

Example: Factorial Function III

Example 17.11 (Factorial function; continued)

Computation for $x = 2$:

	<i>PC</i>	<i>DS</i>	<i>PS</i>
1 : CALL(17,0,1);	1	ϵ	0 : 0 : 0 : 2
2 : JMP(0);	17	ϵ	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
3 : LOAD(2,1);	18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
4 : LIT(1);	19	ϵ	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5 : GT;	3	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6 : JFALSE(16);	4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7 : LOAD(1,1);	5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8 : LOAD(2,1);	6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9 : MULT;	7	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10 : STORE(1,1);	8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11 : LOAD(2,1);			
12 : LIT(1);			
13 : SUB;			
14 : STORE(2,1);			
15 : CALL(3,1,0);			
16 : RET;			
17 : LIT(1);			
18 : STORE(0,1);			
19 : CALL(3,0,0);			
20 : LOAD(0,1);			
21 : STORE(1,1);			
22 : RET;			

Example: Factorial Function III

Example 17.11 (Factorial function; continued)

Computation for $x = 2$:

	<i>PC</i>	<i>DS</i>	<i>PS</i>
1 : CALL(17,0,1);	1	ϵ	0 : 0 : 0 : 2
2 : JMP(0);	17	ϵ	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
3 : LOAD(2,1);	18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
4 : LIT(1);	19	ϵ	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5 : GT;	3	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6 : JFALSE(16);	4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7 : LOAD(1,1);	5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8 : LOAD(2,1);	6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9 : MULT;	7	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10 : STORE(1,1);	8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11 : LOAD(2,1);	9	1 : 2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
12 : LIT(1);			
13 : SUB;			
14 : STORE(2,1);			
15 : CALL(3,1,0);			
16 : RET;			
17 : LIT(1);			
18 : STORE(0,1);			
19 : CALL(3,0,0);			
20 : LOAD(0,1);			
21 : STORE(1,1);			
22 : RET;			

Example: Factorial Function III

Example 17.11 (Factorial function; continued)

Computation for $x = 2$:

	<i>PC</i>	<i>DS</i>	<i>PS</i>
1 : CALL(17,0,1);	1	ϵ	0 : 0 : 0 : 2
2 : JMP(0);	17	ϵ	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
3 : LOAD(2,1);	18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
4 : LIT(1);	19	ϵ	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5 : GT;	3	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6 : JFALSE(16);	4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7 : LOAD(1,1);	5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8 : LOAD(2,1);	6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9 : MULT;	7	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10 : STORE(1,1);	8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11 : LOAD(2,1);	9	1 : 2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
12 : LIT(1);	10	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
13 : SUB;			
14 : STORE(2,1);			
15 : CALL(3,1,0);			
16 : RET;			
17 : LIT(1);			
18 : STORE(0,1);			
19 : CALL(3,0,0);			
20 : LOAD(0,1);			
21 : STORE(1,1);			
22 : RET;			

Example: Factorial Function III

Example 17.11 (Factorial function; continued)

Computation for $x = 2$:

	<i>PC</i>	<i>DS</i>	<i>PS</i>
1 : CALL(17,0,1);	1	ϵ	0 : 0 : 0 : 2
2 : JMP(0);	17	ϵ	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
3 : LOAD(2,1);	18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
4 : LIT(1);	19	ϵ	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5 : GT;	3	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6 : JFALSE(16);	4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7 : LOAD(1,1);	5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8 : LOAD(2,1);	6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9 : MULT;	7	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10 : STORE(1,1);	8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11 : LOAD(2,1);	9	1 : 2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
12 : LIT(1);	10	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
13 : SUB;	11	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
14 : STORE(2,1);			
15 : CALL(3,1,0);			
16 : RET;			
17 : LIT(1);			
18 : STORE(0,1);			
19 : CALL(3,0,0);			
20 : LOAD(0,1);			
21 : STORE(1,1);			
22 : RET;			

Example: Factorial Function III

Example 17.11 (Factorial function; continued)

Computation for $x = 2$:	<i>PC</i>	<i>DS</i>	<i>PS</i>
1 : CALL(17,0,1);	1	ϵ	0 : 0 : 0 : 2
2 : JMP(0);	17	ϵ	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
3 : LOAD(2,1);	18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
4 : LIT(1);	19	ϵ	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5 : GT;	3	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6 : JFALSE(16);	4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7 : LOAD(1,1);	5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8 : LOAD(2,1);	6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9 : MULT;	7	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10 : STORE(1,1);	8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11 : LOAD(2,1);	9	1 : 2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
12 : LIT(1);	10	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
13 : SUB;	11	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
14 : STORE(2,1);	12	2	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
15 : CALL(3,1,0);			
16 : RET;			
17 : LIT(1);			
18 : STORE(0,1);			
19 : CALL(3,0,0);			
20 : LOAD(0,1);			
21 : STORE(1,1);			
22 : RET;			

Example: Factorial Function III

Example 17.11 (Factorial function; continued)

Computation for $x = 2$:

	<i>PC</i>	<i>DS</i>	<i>PS</i>
1 : CALL(17,0,1);	1	ϵ	0 : 0 : 0 : 2
2 : JMP(0);	17	ϵ	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
3 : LOAD(2,1);	18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
4 : LIT(1);	19	ϵ	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5 : GT;	3	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6 : JFALSE(16);	4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7 : LOAD(1,1);	5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8 : LOAD(2,1);	6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9 : MULT;	7	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10 : STORE(1,1);	8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11 : LOAD(2,1);	9	1 : 2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
12 : LIT(1);	10	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
13 : SUB;	11	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
14 : STORE(2,1);	12	2	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
15 : CALL(3,1,0);	13	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
16 : RET;			
17 : LIT(1);			
18 : STORE(0,1);			
19 : CALL(3,0,0);			
20 : LOAD(0,1);			
21 : STORE(1,1);			
22 : RET;			

Example: Factorial Function III

Example 17.11 (Factorial function; continued)

Computation for $x = 2$:

	<i>PC</i>	<i>DS</i>	<i>PS</i>
1 : CALL(17,0,1);	1	ϵ	0 : 0 : 0 : 2
2 : JMP(0);	17	ϵ	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
3 : LOAD(2,1);	18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
4 : LIT(1);	19	ϵ	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5 : GT;	3	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6 : JFALSE(16);	4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7 : LOAD(1,1);	5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8 : LOAD(2,1);	6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9 : MULT;	7	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10 : STORE(1,1);	8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11 : LOAD(2,1);	9	1 : 2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
12 : LIT(1);	10	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
13 : SUB;	11	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
14 : STORE(2,1);	12	2	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
15 : CALL(3,1,0);	13	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
16 : RET;	14	1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
17 : LIT(1);			
18 : STORE(0,1);			
19 : CALL(3,0,0);			
20 : LOAD(0,1);			
21 : STORE(1,1);			
22 : RET;			

Example: Factorial Function III

Example 17.11 (Factorial function; continued)

Computation for $x = 2$:

	<i>PC</i>	<i>DS</i>	<i>PS</i>
1 : CALL(17,0,1);	1	ϵ	0 : 0 : 0 : 2
2 : JMP(0);	17	ϵ	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
3 : LOAD(2,1);	18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
4 : LIT(1);	19	ϵ	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5 : GT;	3	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6 : JFALSE(16);	4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7 : LOAD(1,1);	5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8 : LOAD(2,1);	6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9 : MULT;	7	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10 : STORE(1,1);	8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11 : LOAD(2,1);	9	1 : 2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
12 : LIT(1);	10	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
13 : SUB;	11	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
14 : STORE(2,1);	12	2	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
15 : CALL(3,1,0);	13	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
16 : RET;	14	1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
17 : LIT(1);	15	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
18 : STORE(0,1);			
19 : CALL(3,0,0);			
20 : LOAD(0,1);			
21 : STORE(1,1);			
22 : RET;			

Example: Factorial Function III

Example 17.11 (Factorial function; continued)

Computation for $x = 2$:

	PC	DS	PS
1 : CALL(17,0,1);	1	ϵ	0 : 0 : 0 : 2
2 : JMP(0);	17	ϵ	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
3 : LOAD(2,1);	18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
4 : LIT(1);	19	ϵ	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5 : GT;	3	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6 : JFALSE(16);	4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7 : LOAD(1,1);	5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8 : LOAD(2,1);	6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9 : MULT;	7	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10 : STORE(1,1);	8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11 : LOAD(2,1);	9	1 : 2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
12 : LIT(1);	10	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
13 : SUB;	11	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
14 : STORE(2,1);	12	2	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
15 : CALL(3,1,0);	13	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
16 : RET;	14	1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
17 : LIT(1);	15	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
18 : STORE(0,1);	3	ϵ	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
19 : CALL(3,0,0);			
20 : LOAD(0,1);			
21 : STORE(1,1);			
22 : RET;			

Example: Factorial Function III

Example 17.11 (Factorial function; continued)

Computation for $x = 2$:

	PC	DS	PS
1 : CALL(17,0,1);	1	ϵ	0 : 0 : 0 : 2
2 : JMP(0);	17	ϵ	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
3 : LOAD(2,1);	18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
4 : LIT(1);	19	ϵ	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5 : GT;	3	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6 : JFALSE(16);	4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7 : LOAD(1,1);	5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8 : LOAD(2,1);	6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9 : MULT;	7	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10 : STORE(1,1);	8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11 : LOAD(2,1);	9	1 : 2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
12 : LIT(1);	10	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
13 : SUB;	11	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
14 : STORE(2,1);	12	2	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
15 : CALL(3,1,0);	13	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
16 : RET;	14	1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
17 : LIT(1);	15	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
18 : STORE(0,1);	3	ϵ	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
19 : CALL(3,0,0);	4	1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
20 : LOAD(0,1);			
21 : STORE(1,1);			
22 : RET;			

Example: Factorial Function III

Example 17.11 (Factorial function; continued)

Computation for $x = 2$:

	<i>PC</i>	<i>DS</i>	<i>PS</i>
1 : CALL(17,0,1);	1	ϵ	0 : 0 : 0 : 2
2 : JMP(0);	17	ϵ	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
3 : LOAD(2,1);	18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
4 : LIT(1);	19	ϵ	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5 : GT;	3	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6 : JFALSE(16);	4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7 : LOAD(1,1);	5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8 : LOAD(2,1);	6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9 : MULT;	7	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10 : STORE(1,1);	8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11 : LOAD(2,1);	9	1 : 2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
12 : LIT(1);	10	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
13 : SUB;	11	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
14 : STORE(2,1);	12	2	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
15 : CALL(3,1,0);	13	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
16 : RET;	14	1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
17 : LIT(1);	15	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
18 : STORE(0,1);	3	ϵ	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
19 : CALL(3,0,0);	4	1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
20 : LOAD(0,1);	5	1 : 1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
21 : STORE(1,1);			
22 : RET;			

Example: Factorial Function III

Example 17.11 (Factorial function; continued)

Computation for $x = 2$:

	<i>PC</i>	<i>DS</i>	<i>PS</i>
1 : CALL(17,0,1);	1	ϵ	0 : 0 : 0 : 2
2 : JMP(0);	17	ϵ	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
3 : LOAD(2,1);	18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
4 : LIT(1);	19	ϵ	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5 : GT;	3	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6 : JFALSE(16);	4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7 : LOAD(1,1);	5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8 : LOAD(2,1);	6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9 : MULT;	7	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10 : STORE(1,1);	8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11 : LOAD(2,1);	9	1 : 2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
12 : LIT(1);	10	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
13 : SUB;	11	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
14 : STORE(2,1);	12	2	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
15 : CALL(3,1,0);	13	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
16 : RET;	14	1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
17 : LIT(1);	15	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
18 : STORE(0,1);	3	ϵ	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
19 : CALL(3,0,0);	4	1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
20 : LOAD(0,1);	5	1 : 1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
21 : STORE(1,1);	6	0	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
22 : RET;			

Example: Factorial Function III

Example 17.11 (Factorial function; continued)

Computation for $x = 2$:

	<i>PC</i>	<i>DS</i>	<i>PS</i>
1 : CALL(17,0,1);	1	ϵ	0 : 0 : 0 : 2
2 : JMP(0);	17	ϵ	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
3 : LOAD(2,1);	18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
4 : LIT(1);	19	ϵ	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5 : GT;	3	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6 : JFALSE(16);	4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7 : LOAD(1,1);	5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8 : LOAD(2,1);	6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9 : MULT;	7	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10 : STORE(1,1);	8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11 : LOAD(2,1);	9	1 : 2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
12 : LIT(1);	10	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
13 : SUB;	11	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
14 : STORE(2,1);	12	2	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
15 : CALL(3,1,0);	13	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
16 : RET;	14	1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
17 : LIT(1);	15	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
18 : STORE(0,1);	3	ϵ	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
19 : CALL(3,0,0);	4	1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
20 : LOAD(0,1);	5	1 : 1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
21 : STORE(1,1);	6	0	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
22 : RET;	16	ϵ	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1

Example: Factorial Function III

Example 17.11 (Factorial function; continued)

Computation for $x = 2$:

	<i>PC</i>	<i>DS</i>		<i>PS</i>
1 : CALL(17,0,1);	1	ϵ		0 : 0 : 0 : 2
2 : JMP(0);	17	ϵ	4 : 3 : 2 : 0 :	0 : 0 : 0 : 2
3 : LOAD(2,1);	18	1	4 : 3 : 2 : 0 :	0 : 0 : 0 : 2
4 : LIT(1);	19	ϵ	4 : 3 : 2 : 1 :	0 : 0 : 0 : 2
5 : GT;	3	ϵ	3 : 2 : 20 :	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6 : JFALSE(16);	4	2	3 : 2 : 20 :	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7 : LOAD(1,1);	5	2 : 1	3 : 2 : 20 :	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8 : LOAD(2,1);	6	1	3 : 2 : 20 :	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9 : MULT;	7	ϵ	3 : 2 : 20 :	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10 : STORE(1,1);	8	1	3 : 2 : 20 :	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11 : LOAD(2,1);	9	1 : 2	3 : 2 : 20 :	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
12 : LIT(1);	10	2	3 : 2 : 20 :	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
13 : SUB;	11	ϵ	3 : 2 : 20 :	4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
14 : STORE(2,1);	12	2	3 : 2 : 20 :	4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
15 : CALL(3,1,0);	13	2 : 1	3 : 2 : 20 :	4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
16 : RET;	14	1	3 : 2 : 20 :	4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
17 : LIT(1);	15	ϵ	3 : 2 : 20 :	4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
18 : STORE(0,1);	3	ϵ	6 : 2 : 16 :	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
19 : CALL(3,0,0);	4	1	6 : 2 : 16 :	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
20 : LOAD(0,1);	5	1 : 1	6 : 2 : 16 :	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
21 : STORE(1,1);	6	0	6 : 2 : 16 :	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
22 : RET;	16	ϵ	6 : 2 : 16 :	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
	16	ϵ	3 : 2 : 20 :	4 : 3 : 2 : 2 : 0 : 0 : 0 : 1

Example: Factorial Function III

Example 17.11 (Factorial function; continued)

Computation for $x = 2$:

	PC	DS	PS
1 : CALL(17,0,1);	1	ϵ	0 : 0 : 0 : 2
2 : JMP(0);	17	ϵ	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
3 : LOAD(2,1);	18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
4 : LIT(1);	19	ϵ	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5 : GT;	3	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6 : JFALSE(16);	4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7 : LOAD(1,1);	5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8 : LOAD(2,1);	6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9 : MULT;	7	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10 : STORE(1,1);	8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11 : LOAD(2,1);	9	1 : 2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
12 : LIT(1);	10	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
13 : SUB;	11	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
14 : STORE(2,1);	12	2	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
15 : CALL(3,1,0);	13	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
16 : RET;	14	1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
17 : LIT(1);	15	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
18 : STORE(0,1);	3	ϵ	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
19 : CALL(3,0,0);	4	1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
20 : LOAD(0,1);	5	1 : 1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
21 : STORE(1,1);	6	0	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
22 : RET;	16	ϵ	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
	16	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
	20	ϵ	4 : 3 : 2 : 2 : 0 : 0 : 0 : 1

Example: Factorial Function III

Example 17.11 (Factorial function; continued)

Computation for $x = 2$:

	PC	DS	PS
1 : CALL(17,0,1);	1	ϵ	0 : 0 : 0 : 2
2 : JMP(0);	17	ϵ	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
3 : LOAD(2,1);	18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
4 : LIT(1);	19	ϵ	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5 : GT;	3	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6 : JFALSE(16);	4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7 : LOAD(1,1);	5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8 : LOAD(2,1);	6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9 : MULT;	7	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10 : STORE(1,1);	8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11 : LOAD(2,1);	9	1 : 2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
12 : LIT(1);	10	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
13 : SUB;	11	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
14 : STORE(2,1);	12	2	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
15 : CALL(3,1,0);	13	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
16 : RET;	14	1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
17 : LIT(1);	15	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
18 : STORE(0,1);	3	ϵ	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
19 : CALL(3,0,0);	4	1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
20 : LOAD(0,1);	5	1 : 1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
21 : STORE(1,1);	6	0	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
22 : RET;	16	ϵ	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
	16	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
	20	ϵ	4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
	21	2	4 : 3 : 2 : 2 : 0 : 0 : 0 : 1

Example: Factorial Function III

Example 17.11 (Factorial function; continued)

Computation for $x = 2$:

	PC	DS	PS
1 : CALL(17,0,1);	1	ϵ	0 : 0 : 0 : 2
2 : JMP(0);	17	ϵ	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
3 : LOAD(2,1);	18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
4 : LIT(1);	19	ϵ	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5 : GT;	3	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6 : JFALSE(16);	4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7 : LOAD(1,1);	5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8 : LOAD(2,1);	6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9 : MULT;	7	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10 : STORE(1,1);	8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11 : LOAD(2,1);	9	1 : 2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
12 : LIT(1);	10	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
13 : SUB;	11	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
14 : STORE(2,1);	12	2	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
15 : CALL(3,1,0);	13	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
16 : RET;	14	1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
17 : LIT(1);	15	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
18 : STORE(0,1);	3	ϵ	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
19 : CALL(3,0,0);	4	1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
20 : LOAD(0,1);	5	1 : 1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
21 : STORE(1,1);	6	0	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
22 : RET;	16	ϵ	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
	16	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
	20	ϵ	4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
	21	2	4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
	22	ϵ	4 : 3 : 2 : 2 : 0 : 0 : 0 : 2

Example: Factorial Function III

Example 17.11 (Factorial function; continued)

Computation for $x = 2$:

	PC	DS	PS
1 : CALL(17,0,1);	1	ϵ	0 : 0 : 0 : 2
2 : JMP(0);	17	ϵ	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
3 : LOAD(2,1);	18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
4 : LIT(1);	19	ϵ	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5 : GT;	3	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6 : JFALSE(16);	4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7 : LOAD(1,1);	5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8 : LOAD(2,1);	6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9 : MULT;	7	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10 : STORE(1,1);	8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11 : LOAD(2,1);	9	1 : 2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
12 : LIT(1);	10	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
13 : SUB;	11	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
14 : STORE(2,1);	12	2	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
15 : CALL(3,1,0);	13	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
16 : RET;	14	1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
17 : LIT(1);	15	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
18 : STORE(0,1);	3	ϵ	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
19 : CALL(3,0,0);	4	1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
20 : LOAD(0,1);	5	1 : 1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
21 : STORE(1,1);	6	0	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
22 : RET;	16	ϵ	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
	16	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
	20	ϵ	4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
	21	2	4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
	22	ϵ	4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
	2	ϵ	0 : 0 : 0 : 2

Example: Factorial Function III

Example 17.11 (Factorial function; continued)

Computation for $x = 2$:

	PC	DS	PS
1 : CALL(17,0,1);	1	ϵ	0 : 0 : 0 : 2
2 : JMP(0);	17	ϵ	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
3 : LOAD(2,1);	18	1	4 : 3 : 2 : 0 : 0 : 0 : 0 : 2
4 : LIT(1);	19	ϵ	4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
5 : GT;	3	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
6 : JFALSE(16);	4	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
7 : LOAD(1,1);	5	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
8 : LOAD(2,1);	6	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
9 : MULT;	7	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
10 : STORE(1,1);	8	1	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
11 : LOAD(2,1);	9	1 : 2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
12 : LIT(1);	10	2	3 : 2 : 20 : 4 : 3 : 2 : 1 : 0 : 0 : 0 : 2
13 : SUB;	11	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
14 : STORE(2,1);	12	2	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
15 : CALL(3,1,0);	13	2 : 1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
16 : RET;	14	1	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
17 : LIT(1);	15	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
18 : STORE(0,1);	3	ϵ	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
19 : CALL(3,0,0);	4	1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
20 : LOAD(0,1);	5	1 : 1	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
21 : STORE(1,1);	6	0	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
22 : RET;	16	ϵ	6 : 2 : 16 : 3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
	16	ϵ	3 : 2 : 20 : 4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
	20	ϵ	4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
	21	2	4 : 3 : 2 : 2 : 0 : 0 : 0 : 1
	22	ϵ	4 : 3 : 2 : 2 : 0 : 0 : 0 : 2
	2	ϵ	0 : 0 : 0 : 2
	0	ϵ	0 : 0 : 0 : 2

- 1 Repetition: Intermediate Code
- 2 Semantics of Procedure and Transfer Instructions
- 3 The Symbol Table
- 4 Translation of Programs
- 5 Translation of Blocks
- 6 Translation of Declarations
- 7 Translation of Commands
- 8 Translation of Expressions
- 9 A Translation Example
- 10 Correctness of the Translation**
- 11 Outlook

Theorem 17.12 (Correctness of translation)

For every $P \in Pgm$, $n \in \mathbb{N}$, and $(z_1, \dots, z_n), (z'_1, \dots, z'_n) \in \mathbb{Z}^n$:

$$\begin{aligned} & \mathfrak{M}[P](z_1, \dots, z_n) = (z'_1, \dots, z'_n) \\ \iff & \llbracket \text{trans}(P) \rrbracket(1, \varepsilon, 0 : 0 : 0 : z_1 : \dots : z_n) = (0, \varepsilon, 0 : 0 : 0 : z'_1 : \dots : z'_n) \end{aligned}$$

Theorem 17.12 (Correctness of translation)

For every $P \in Pgm$, $n \in \mathbb{N}$, and $(z_1, \dots, z_n), (z'_1, \dots, z'_n) \in \mathbb{Z}^n$:

$$\begin{aligned} & \mathfrak{M}[P](z_1, \dots, z_n) = (z'_1, \dots, z'_n) \\ \iff & \llbracket \text{trans}(P) \rrbracket(1, \varepsilon, 0 : 0 : 0 : z_1 : \dots : z_n) = (0, \varepsilon, 0 : 0 : 0 : z'_1 : \dots : z'_n) \end{aligned}$$

Proof.

see M. Mohnen: *A Compiler Correctness Proof for the Static Link Technique by means of Evolving Algebras*, Fundamenta Informaticae 29(3), 1997, pp. 257–303



- 1 Repetition: Intermediate Code
- 2 Semantics of Procedure and Transfer Instructions
- 3 The Symbol Table
- 4 Translation of Programs
- 5 Translation of Blocks
- 6 Translation of Declarations
- 7 Translation of Commands
- 8 Translation of Expressions
- 9 A Translation Example
- 10 Correctness of the Translation
- 11 Outlook

- ① More about **code generation**:
 - Handling of procedure parameters
 - Static & dynamic data structures
 - Compiler backend (register allocation, instruction selection & placement, ...)
 - Code analysis & optimization
 - ⇒ *Static Program Analysis* in Winter semester

- ① More about **code generation**:
 - Handling of procedure parameters
 - Static & dynamic data structures
 - Compiler backend (register allocation, instruction selection & placement, ...)
 - Code analysis & optimization
 - ⇒ *Static Program Analysis* in Winter semester
- ② June 27/28, July 4: **Computational Differentiation** [Naumann]
- ③ July 11: **preparation of exam** (question time)
- ④ July 12: **1st exam**