Pcode

(In the following, explicit references to pointers are omitted. Actually, the stack consists entirely of pointers.)

LoadL x
loads the lvalue of x onto the stack.

LoadR x
loads the rvalue of x onto the stack.

LoadS x
creates a Pal string representation of 'x' and loads this onto the top of the stack.

LoadN type x
creates a node of the specified type (NUMBER or REAL) and loads this onto the top of the stack.

LoadJ
creates a JJ node which is loaded onto the stack. (A JJ node is an rvalue which is of type environment (not to be confused with an environment node, which is not an rvalue). LoadJ is produced only by valof; *res*

is subsequently declared, and the JJ node becomes its value. There is no way for the Pal user to access JJ nodes created in this way; however, JJ nodes created by use of the library function SaveENV can be accessed.)

Result (return from a result block)

1. The current value of *res* is obtained (it is required to be a JJ node) and the values of C, S and E contained in this node (the ones existing when the node was created) are restored.
2. Return is executed (see below).

(Result is produced only by res.)

Return (return from a function or block)

1. The values of C and E contained in the current dump (words 4-6 of the current STACK node) are restored.
2. A new STACK node is created which is identical to the one pointed to in the current dump; S is set to this new node.
3. StackP is set to the value indicated in this new STACK node.
4. The content of the top of the previous STACK node ("the return value") is placed on top of the new stack.

True, False, Nil, Dummy

loads the appropriate node onto the stack.

Formlvalue

pops the top of the stack, creates an LVALUE node pointing to this value, and places the LVALUE node on top of the stack.

Formrvalue

pops the top of stack (assumed to be an LVALUE node), obtains its value, and places this node on top of the stack.

Formclosure Ln

creates a CLOSURE node and places it on top of the stack. The CLOSURE node contains a pointer to the current environment and the machine address specified by Ln.

Binop (where Binop = Mult, Div, Plus, Minus, Power, Eq, Ne, Ls, Gr, Le, Ge, Logand, Logor or Aug)

the top two elements of the stack are removed, the operator is applied to these elements, and the result is placed on the stack.

Unop (where Unop = Pos, Neg or Not)

the top element of the stack is removed, the operator is applied to this element, and the result is placed on the stack.

Tuple n

the top n elements of the stack are removed and combined into an n-tuple which is placed on the stack. The top member of the stack becomes the first member of the tuple.

Members n

the top element of the stack (assumed to be an n-tuple) is removed and its n members are placed on the stack, last member first (hence, the first member of the tuple becomes the top member of the stack). (Members is used only for handling variables declared in a simultaneous definition.)

Apply (apply a function or a tuple)

The top element of the stack is removed.

1. If it is a tuple, the next element of the stack is also removed, the tuple is applied to this element (assumed to be a number), and the result is placed on the stack.
2. If the element is a BASICFN node, the appropriate BCPL subroutine is executed.
3. If the element is a CLOSURE node, the current value of C is placed in OldC, C is set to the value specified in the CLOSURE node, and the CLOSURE node is returned to the top of the stack (I can see no reason for this last action). (The first Pocode instruction of the subroutine is always Save.)

Blocklink Ln (set up link for entering a block)

(Blocklink makes it possible to use the same code for entering and leaving a block as is used for entering and leaving a subroutine.)

1. A NIL node is placed on top of the stack (corresponding to the CLOSURE node which is on the top of the stack when a CLOSURE is Apply'd).
2. The location Ln is placed in OldC (this is the location of the first instruction following the block).
3. The current value of E is placed in a temporary location.

(Blocklink is always followed by Save.)

Reslink Ln (set up link for entering a result block)

1. An LVALUE node having nil as its rvalue is created and placed on the top of the stack. (I can see no reason for this action.)
2. Blocklink(Ln) is executed.

Save Lm

creates a new STACK node and makes this the current stack node. The length of the node is Lm+6. The content of OldC (see Apply and Blocklink) is placed in the fourth word of this node, and the old values of S and E are placed in the fifth and sixth words. The next-to-the-top element of the old stack is placed on top of the stack (this element is nil if a block is being entered; otherwise it is the argument of the function being applied). A new value of E is obtained from either the CLOSURE node or, in the case of a block, from a temporary location (in the latter case, this value is identical to the former value of E). The old value of StackP is saved in the third word of the old STACK node.

Testempty

used to make sure that the argument of a function of no arguments is indeed nil.

Declname x

results in adding to the environment an ENV node for the variable x. The top element of the stack (assumed to be an LVALUE node) is removed from the stack and becomes the lvalue of x.

Declnames n $x_1 \cdots x_n$ $(n > 1)$

results in adding to the environment ENV nodes for the variables $x_1, \ldots, x_n$. The top element of the stack (which must be an n-tuple) is removed; the elements of this tuple become the lvalues of $x_1, \ldots, x_n$.

Decllabel x Ln

results in adding to the environment an ENV node for the variable x. LVALUE, LABEL and STACK nodes are also created. The LABEL node becomes the rvalue of x and contains the location specified by Ln.

SetlabEs n

The n preceding statements having all been Decllabel, this statement results in storing the current value of E in each of the LABEL nodes which have just been created. (This environment differs from the ones existing at the time these nodes were created in that it contains the declarations of all these labels.)

Jump Ln

C is set to the location specified by Ln. (used for internally generated transfers of control)

JumpF Ln

The top element of the stack is removed and, if it is false, C is set to the location specified by Ln.

Goto

The top element of the stack is removed and, if it is a label, it is gone to: values of C and E obtained from the LABEL node, a new STACK node having the appropriate length and contents is created, and S is set to this node.

Lose1

The top element of the stack is removed. (Lose1 is generated by a semi-colon in the source program.)

Update n

The top two elements of the stack are removed and

1. if n=1, the top element becomes the rvalue of the second element (an LVALUE node).
2. if n>1, the n rvalues of the top element (which must be an n-tuple) become rvalues of the second element.

The DUMMY node is placed on top of the stack.

Recursion

The Pal program let rec x = ~ in ~

is translated into

LoadE		*
Loadguess		*
Declname	x	*
~		
Initname	x	*
LoadR	x	*
FormIvalue		*
RestoreE1		*
Blocklink	Lm	
Save	Ln	
Declname	x	
~		

The asterisked statements are used to set up a local environment in which f is evaluated. Those statements which have not been previously discussed are discussed on the next page.

LoadE

loads the current value of E onto the stack.

Loadguess

loads a GUESS node onto the stack. This node is used as a temporary value of the recursive variable.

Initname x

removes the top element of the stack and makes this the value of x.

Initnames n $x_1 \cdots x_n$ $(n > 1)$

removes the top element of the stack (which must be an n-tuple); the elements of this tuple become the values of $x_1, \ldots, x_n$.

RestoreE1

removes the next to the top element of the stack (this is the value of E which was saved by LoadE) and makes this the current value of E.