Team 04
Haohan Jiang, g3jiangh
Maria Yancheva, c2yanche
Timo Vink, c4vinkti
Chandeep Singh, g2singh

1 Program 3

As described above, we start with the HALT instruction at address 0, which will be used as our return address for the 'main procedure'.

```
0    HALT
```

Next we need to put the activation record on the stack and set the display register to point to it. The activation record contains the return address 0, space to save a display register, and space for local variables.

```
     # Set display register
1    PUSHMT
2    SETD       0

      # Create activation record: return address, dynamic link, display[M], 8 params+vars
3    PUSH       0
4    PUSH       UNDEFINED
5    PUSH       UNDEFINED
6    PUSH       UNDEFINED
7    PUSH       8
8    DUPN
```

The first line requiring code generation is line 3-29. Before calling procedure Q, we save the display data for lexical level 1. Since the main program does not have a return value, it is equivalent to a procedure (i.e., its display[M] entry is the third one in its activation record stack).

```
      # Get address of display[M] entry in the activation record of main program
9     ADDR       0       2

      # Get display data for lexical level 1, and store it in the main program
10    ADDR       1       0
11    STORE
```

Allocate space for control items in activation record of Q: return address, dynamic link and display:

```
      # return_addr_Q
12    PUSH      242
13    ADDR      0       0
14    PUSH      UNDEFINED
```

Next, update the display for lexical level 1:

```
15    PUSHMT
16    PUSH          2
17    SUB
18    SETD          1
```

Evaluate argument expressions, write them to activation record, and branch to the procedure body code.

Q: argument 1

```
      # Not p
19    1
20    ADDR      0       7
21    LOAD
22    SUB

      # Or q
23    ADDR      0       8
24    LOAD
25    OR
```

Q: argument 2. Execute function call to F.

F (call 1): store display data for lexical level 1 within caller.

```
26    ADDR      1       2
27    ADDR      1       0
28    STORE
```

F (call 1): allocate space for return value, return address, dynamic link and display.

```
29    PUSH      UNDEFINED
      # return_addr_for_F1
30    PUSH      88
31    ADDR      1       0
32    PUSH      UNDEFINED
```

F (call 1): update the display for lexical level 1 to point to current activation record.

```
33    PUSHMT
34    PUSH      3
35    SUB
36    SETD      1
```

F (call 1): evaluate parameter expressions. Argument 1: execute function call to F.

F (call 2): store display data for lexical level 1 within caller:

```
37    ADDR      1       3
38    ADDR      1       0
39    STORE
```

F (call 2): allocate space for return value, return address, dynamic link and display.

```
40    PUSH      UNDEFINED
      # return_addr_for_F2
41    PUSH      82
42    ADDR      1       0
43    PUSH      UNDEFINED
```

F (call 2): update the display for lexical level 1 to point to current activation record.

```
44    PUSHMT
45    PUSH      3
46    SUB
47    SETD      1
```

F (call 2): evaluate parameter expressions. Argument 1: b, argument 2: p. Both exist in lexical level 0.

```
48    ADDR      0       4
49    LOAD
50    ADDR      0       7
51    LOAD
```

F (call 2): branch to function entrance code.

```
      # addr_F_entrance_code
52    PUSH      54
53    BR
```

F entrance code: allocate space for parameters and identifiers.

```
54    PUSH      UNDEFINED
55    PUSH      2
56    DUPN

      # F body code
57    ADDR      1       5
58    LOAD
      # branch_false_addr
59    PUSH      68
60    BF
      # True condition code: return m+b
61    ADDR      1       4
62    LOAD
63    ADDR      0       4
64    LOAD
65    ADD
      # addr_F_epiloguecode
66    PUSH      75
67    BR

      # False condition code: return c-m
68    ADDR      0       5
```

```
69    LOAD
70    ADDR      1       4
71    LOAD
72    SUB
      # addr_F_epiloguecode
73    PUSH      75
74    BR
```

F epilogue code: pop all params + identifiers, and restore the display data from parent's activation record. Finally, the return address is on the top of the stack, so simply branch to it.

```
75    PUSH      2
76    POPN
77    POP
78    PUSH      3
79    LOAD
80    SETD      1
81    BR
```

F (call 1): argument 2 (not q).

```
82    PUSH      1
83    ADDR      0       8
84    LOAD
85    SUB
```

F (call 1): branch to function entrance code.

```
      # addr_F_entrance_code
86    PUSH      54
87    BR
```

Q: argument 3. Execute anonymous function call.
Anonymous function: store current display[M] into the caller (Q).

```
88    ADDR      1       2
89    ADDR      2       0
90    STORE
```

Anonymous function: allocate space for return value, return address, dynamic link and display.

```
91    PUSH      UNDEFINED
      # return_addr_anon
92    PUSH      146
93    ADDR      1       0
94    PUSH      UNDEFINED
```

Anonymous function: update display.

```
95    PUSHMT
96    PUSH      3
97    SUB
98    SETD      2
```

Anonymous function: no parameter expressions to evaluate. Execute body code. First statement invokes a call to procedure P.

P: store current display[M] into the caller (anon).

```
99    ADDR      2       3
100   ADDR      1       0
101   STORE
```

P: allocate space for return address, dynamic link and display.

```
      # return_addr_P
102   PUSH      133
103   ADDR      2       0
104   PUSH      UNDEFINED
```

P: update display.

```
105   PUSHMT
106   PUSH      2
107   SUB
108   SETD      1
```

P: no parameter expressions to evaluate. Branch to procedure entrance code and body.

```
      # addr_P_entrancecode
109   PUSH      111
110   BR
```

P: entrance code. Allocate space for identifiers. Then execute body statements.

```
111   PUSH      UNDEFINED
112   PUSH      2
113   DUPN

      # P body code
114   ADDR      0       7
115   LOAD
      # addr_fwd
116   PUSH      120
117   BF
      # True condition code. addr_epilogue_P:
118   PUSH      126
119   BR

      # Assignment e <= a
120   ADDR      1       3
121   ADDR      0       3
122   LOAD
123   STORE

      # Return (branch to addr_epilogue_P)
124   PUSH      126
125   BR
```

P: epilogue. Pop all identifiers off the stack. Pop display. Restore display from caller. Then branch to return address.

```
126   PUSH      2
127   POPN
128   POP
129   PUSH      3
130   LOAD
131   SETD      1
132   BR
```

Anonymous function: return statement.

```
133   PUSH      1
134   ADDR      0       7
135   LOAD
136   ADDR      0       8
137   EQ
138   SUB
      # addr_epilogue_anon
139   PUSH      141
140   BR
```

Anonymous function: epilogue. Pop display, restore display, branch to return address.

```
141   POP
142   PUSH      2
143   LOAD
144   SETD      2
145   BR
```

Q: branch to function entrance code.

```
      # addr_entrancecode_Q
146   PUSH      148
147   BR
```

Q: entrance code. Allocate space for params and identifiers. Then execute body statements.

```
148   PUSH      UNDEFINED
149   PUSH      6
150   DUPN

      # Now call function F.
```

F (call 3): save current display in caller (Q).

```
151   ADDR      1       2
152   ADDR      1       0
153   STORE
```

F (call 3): allocate space for return value, return address, dynamic link and display.

```
154   PUSH      UNDEFINED
      # return_addr_for_F3
155   PUSH      230
156   ADDR      1        0
157   PUSH      UNDEFINED
```

F (call 3): update display.

```
158   PUSHMT
159   PUSH      3
160   SUB
161   SETD      1
```

F (call 3): evaluate parameter expressions. Argument 1: t - n + a.

```
162   ADDR      1        6
163   LOAD
164   ADDR      1        4
165   LOAD
166   SUB
167   ADDR      0        3
168   LOAD
169   ADD
```

F (call 3): argument 2.

```
170   PUSH      1
```

At this point, need to execute function G. Save current display in caller (F).

```
151   ADDR      1        3
152   ADDR      2        0
153   STORE
```

G: allocate space for return value, return address, dynamic link and display.

```
154   PUSH      UNDEFINED
      # return_addr_G
155   PUSH      224
156   ADDR      1        0
157   PUSH      UNDEFINED
```

G: update display.

```
158   PUSHMT
159   PUSH      3
160   SUB
161   SETD      2
```

G: no parameters to evaluate. Branch to function entrance code.

```
      # addr_entrancecode_G
162   PUSH      164
163   BR
```

G: entrance code. Allocate space for identifiers. Then execute body code.

```
164   PUSH      UNDEFINED
165   PUSH      2
166   DUPN

      # Body of G: execute anonymous function.
```

Anonymous function (call 2): save current display into caller.

```
167   ADDR      2       3
168   ADDR      3       0
169   STORE
```

Anonymous function (call 2): allocate space for return value, return address, dynamic link and display.

```
170   PUSH      UNDEFINED
      # return_addr_anon2
171   PUSH      215
172   ADDR      2       0
173   PUSH      UNDEFINED
```

Anonymous function (call 2): update display.

```
174   PUSHMT
175   PUSH      3
176   SUB
177   SETD      3
```

Anonymous function (call 2): no parameters to evaluate. Execute function entrance code and body statements.

```
178   PUSH      UNDEFINED
179   PUSH      2
180   DUPN
181   ADDR      3       5
182   ADDR      0       5
183   STORE

      # Call procedure P.
```

P (call 2): store current display[M] into the caller (anon 2).

```
184    ADDR      3       3
185   ADDR      1       0
186   STORE
```

P (call 2): allocate space for return address, dynamic link and display.

```
      # return_addr_P
187   PUSH      196
188   ADDR      3       0
189   PUSH      UNDEFINED
```

P: update display.

```
190   PUSHMT
191   PUSH      2
192   SUB
193   SETD      1
```

P: no parameter expressions to evaluate. Branch to procedure entrance code and body.

```
      # addr_P_entrancecode
194   PUSH      111
195   BR
```

Anonymous function (call 2): execute return statement.

```
196   ADDR      3       5
197   LOAD
198   ADDR      2       4
199   LOAD
200   ADD
201   ADDR      1       8
202   LOAD
203   SUB
204   PUSH      12
205   LT
      # addr_epilogue_anon2
206   PUSH      208
207   BR
```

Anonymous function (call 2): epilogue. Clean up allocated space. Pop display. Restore display. Then branch to return address.

```
208   PUSH      2
209   POPN
210   POP
211   PUSH      3
212   LOAD
213   SETD      3
214   BR
```

G: at this point, the return expression (returned by the anonymous function) is at the top of the stack. Now execute the return statement.

```
      # addr_epilogue_G
215   PUSH      217
216   BR
```

G: epilogue. Clean up allocated space. Pop display. Restore display. Then branch to return address.

```
217   PUSH      2
218   POPN
219   POP
220   PUSH      3
221   LOAD
222   SETD      2
223   BR
```

F (call 3): argument 2 processing. Right now at top of stack we have the return value of G.

```
      # 1 - return value of G => !G
224   SUB
225   ADDR      0       9
226   LOAD
227   OR
```

F (call 3): branch to function entrance code.

```
      # addr_entrancecode_F
228   PUSH      54
229   BR
```

Q: at this point we have the return value of F at the top of the stack. Print it out, then print out a newline (skip), which is ASCII character code 10.

```
      # Print out return value of F
230   PRINTI
      # Print out newline (skip)
231   PUSH      10
232   PRINTC
```

Q: the body has been executed. Now go to epilogue code.

```
      # addr_epilogue_Q
233   PUSH      235
234   BR
```

Q: epilogue. Clean up allocated space. Pop display. Restore display. Branch to return address.

```
235   PUSH      6
236   POPN
237   POP
238   PUSH      2
239   LOAD
240   SETD      1
241   BR
```

Main program: we have finished executing all body statements. Now branch to epilogue code.

```
      # addr_epilogue_main
242   PUSH      244
243   BR
```

Main program: epilogue. Pop identifiers, pop display. Branch to return address.

```
244   PUSH      8
245   POPN

      # pop display and dynamic link words
246   POP
247   POP

      # Branch to return address
248   ADDR        0       0
249   LOAD
250   BR
```