

The Move instruction operates on the Register Machine, and *_copies_* data from one register into the other.

Moves can be from one register to another, or we can move a literal value into a register.

The representation of literal values isn't important, so an implementation may copy the actual value or just a heap pointer.

```

10  ┌────────────────── MODULE LAM_ISA_Move ───────────────────┐
11  EXTENDS Naturals, Sequences, TLC

13  CONSTANTS InstrCount, RegisterCount, Nil, Literals

15  RegisterKinds  $\triangleq$  { "local", "global" }
16  RegisterIdx  $\triangleq$  (0 .. RegisterCount)
17  ValueKinds  $\triangleq$  RegisterKinds  $\cup$  { "literal" }
18  Values  $\triangleq$  RegisterIdx  $\cup$  Literals

20  Registers  $\triangleq$  RegisterKinds  $\times$  RegisterIdx

22  --algorithm LAM_ISA_Move
23  variables
24    registers = [r  $\in$  Registers  $\mapsto$  Nil],
25    current_count = InstrCount,
26    current_move = Nil
27    ;
29  define
31    AllRegistersAreValid  $\triangleq$   $\vee$  current_move = Nil
32                         $\vee$   $\wedge$  current_move.dst[1]  $\in$  RegisterKinds
33                         $\wedge$  current_move.dst[2]  $\in$  RegisterIdx
34                         $\wedge$   $\vee$   $\wedge$  current_move.src[1]  $\in$  RegisterKinds
35                         $\wedge$  current_move.src[2]  $\in$  RegisterIdx
36                         $\vee$   $\wedge$  current_move.src[1]  $\in$  { "literal" }
37                         $\wedge$  current_move.src[2]  $\in$  Literals

39    TypeInvariant  $\triangleq$   $\wedge$  current_count  $\in$  Nat
40                         $\wedge$  AllRegistersAreValid

43  end define ;

45  procedure perform_move(move) begin
46    PerformMove:
47      if move.src[1] = "literal" then registers[move.dst] := move.src[2]
48      else registers[move.dst] := registers[move.src]
49      end if ;
50      return ;
51  end procedure ;

```

```

53 begin
54   Run:
55   while current_count > 0 do
56     current_count := current_count - 1 ;
57     with src ∈ Registers,
58         dst ∈ Registers,
59         use_lit ∈ {TRUE, FALSE},
60         lit ∈ Literals do
61       Lets assert that the last move we did was actually carried out!
62       assert (
63         ∨ current_move = Nil
64         ∨ ∧ current_move.src[1] = "literal"
65           ∧ registers[current_move.dst] = current_move.src[2]
66           ∨ registers[current_move.dst] = registers[current_move.src]
67         ) = TRUE ;
68       if use_lit then
69         current_move := [src ↦ ⟨"literal", lit⟩, dst ↦ dst] ;
70       else
71         current_move := [src ↦ src, dst ↦ dst] ;
72       end if ;
73       call perform_move(current_move) ;
74     end with ;
75   end while ;
76 end algorithm ;
77 BEGIN TRANSLATION (chksum(pcal) = "abda9f12" ∧ chksum(tla) = "99d227dd")
78 CONSTANT defaultInitValue
79 VARIABLES registers, current_count, current_move, pc, stack

81 define statement
82 AllRegistersAreValid  $\triangleq$  ∨ current_move = Nil
83                       ∨ ∧ current_move.dst[1] ∈ RegisterKinds
84                       ∧ current_move.dst[2] ∈ RegisterIdx
85                       ∧ ∨ ∧ current_move.src[1] ∈ RegisterKinds
86                       ∧ current_move.src[2] ∈ RegisterIdx
87                       ∨ ∧ current_move.src[1] ∈ {"literal"}
88                       ∧ current_move.src[2] ∈ Literals

90 TypeInvariant  $\triangleq$  ∧ current_count ∈ Nat
91                  ∧ AllRegistersAreValid

93 VARIABLE move

95 vars  $\triangleq$  ⟨registers, current_count, current_move, pc, stack, move⟩

97 Init  $\triangleq$  Global variables
98          ∧ registers = [r ∈ Registers ↦ Nil]

```

```

99       $\wedge$  current_count = InstrCount
100      $\wedge$  current_move = Nil
101     Procedure perform_move
102      $\wedge$  move = defaultInitValue
103      $\wedge$  stack =  $\langle \rangle$ 
104      $\wedge$  pc = "Run"

106   PerformMove  $\triangleq$   $\wedge$  pc = "PerformMove"
107      $\wedge$  IF move.src[1] = "literal"
108       THEN  $\wedge$  registers' = [registers EXCEPT ![move.dst] = move.src[2]]
109       ELSE  $\wedge$  registers' = [registers EXCEPT ![move.dst] = registers[move.src]]
110      $\wedge$  pc' = Head(stack).pc
111      $\wedge$  move' = Head(stack).move
112      $\wedge$  stack' = Tail(stack)
113      $\wedge$  UNCHANGED  $\langle$ current_count, current_move $\rangle$ 

115   perform_move  $\triangleq$  PerformMove

117   Run  $\triangleq$   $\wedge$  pc = "Run"
118      $\wedge$  IF current_count > 0
119       THEN  $\wedge$  current_count' = current_count - 1
120        $\wedge$   $\exists$  src  $\in$  Registers :
121          $\exists$  dst  $\in$  Registers :
122            $\exists$  use_lit  $\in$  {TRUE, FALSE} :
123              $\exists$  lit  $\in$  Literals :
124                $\wedge$  Assert((
125                  $\vee$  current_move = Nil
126                  $\vee$   $\wedge$  current_move.src[1] = "literal"
127                    $\wedge$  registers[current_move.dst] = current_move.src[2]
128                    $\vee$  registers[current_move.dst] = registers[current_move.src]
129                 ) = TRUE,
130               "Failure of assertion at line 62, column 9.")
131              $\wedge$  IF use_lit
132               THEN  $\wedge$  current_move' = [src  $\mapsto$   $\langle$ "literal", lit $\rangle$ , dst  $\mapsto$  dst]
133               ELSE  $\wedge$  current_move' = [src  $\mapsto$  src, dst  $\mapsto$  dst]
134              $\wedge$   $\wedge$  move' = current_move'
135              $\wedge$  stack' =  $\langle$ [procedure  $\mapsto$  "perform_move",
136                       pc  $\mapsto$  "Run",
137                       move  $\mapsto$  move] $\rangle$ 
138                        $\circ$  stack
139              $\wedge$  pc' = "PerformMove"
140           ELSE  $\wedge$  pc' = "Done"
141            $\wedge$  UNCHANGED  $\langle$ current_count, current_move, stack, move $\rangle$ 
142      $\wedge$  UNCHANGED registers

144   Allow infinite stuttering to prevent deadlock on termination.

```

145 $Terminating \triangleq pc = \text{"Done"} \wedge \text{UNCHANGED } vars$

147 $Next \triangleq perform_move \vee Run$
148 $\vee Terminating$

150 $Spec \triangleq Init \wedge \Box[Next]_{vars}$

152 $Termination \triangleq \Diamond(pc = \text{"Done"})$

154 **END TRANSLATION**

157

\ * Modification History
\ * Last modified Sat Dec 19 14:49:24 CET 2020 by osteru
\ * Created Sat Dec 19 11:40:24 CET 2020 by osteru