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 ≜ {"local", "global"}
16 RegisterIdx ≜ (0 .. RegisterCount)
17 ValueKinds ≜ RegisterKinds ∪ {"literal"}
18 Values ≜ RegisterIdx ∪ Literals

20 Registers ≜ RegisterKinds × RegisterIdx

22   --algorithm LAM_ISA_Move
23 variables
24   registers = [r ∈ Registers ↦ Nil],
25   current_count = InstrCount,
26   current_move = Nil
27   ;

29 define

31   AllRegistersAreValid ≜ ∨ current_move = Nil
32                          ∨ ∧ current_move.dst[1] ∈ RegisterKinds
33                            ∧ current_move.dst[2] ∈ RegisterIdx
34                            ∧ ∨ ∧ current_move.src[1] ∈ RegisterKinds
35                                ∧ current_move.src[2] ∈ RegisterIdx
36                              ∨ ∧ current_move.src[1] ∈ {"literal"}
37                                ∧ current_move.src[2] ∈ Literals

39   TypeInvariant ≜ ∧ current_count ∈ Nat
40                   ∧ 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 ;
```

```
begin
  Run:
    while current_count > 0 do
      current_count := current_count − 1 ;
      with src ∈ Registers,
           dst ∈ Registers,
           use_lit ∈ {TRUE, FALSE},
           lit ∈ Literals do
        Lets assert that the last move we did was actually carried out!
        assert (
                ∨ current_move = Nil
                ∨ ∧ current_move.src[1] = "literal"
                  ∧ registers[current_move.dst] = current_move.src[2]
                ∨ registers[current_move.dst] = registers[current_move.src]
               ) = TRUE ;
        if use_lit then
          current_move := [src ↦ ⟨"literal", lit⟩,  dst ↦ dst] ;
         else
          current_move := [src ↦ src,  dst ↦ dst] ;
        end if ;
        call perform_move(current_move) ;
      end with ;
    end while ;
end algorithm  ;
 BEGIN TRANSLATION (chksum(pcal) = "63cf1f" ∧ chksum(tla) = "8d3d4b41")
CONSTANT defaultInitValue
VARIABLES global_registers, local_registers, current_count, current_move, pc,
          stack

 define statement
AllRegistersAreValid ≜ ∨ current_move = Nil
                       ∨ ∧ current_move.dst[1] ∈ RegisterKinds
                         ∧ current_move.dst[2] ∈ RegisterIdx
                         ∧ ∨ ∧ current_move.src[1] ∈ RegisterKinds
                               ∧ current_move.src[2] ∈ RegisterIdx
                           ∨ ∧ current_move.src[1] ∈ {"literal"}
                               ∧ current_move.src[2] ∈ Literals

TypeInvariant ≜ ∧ current_count ∈ Nat
                ∧ AllRegistersAreValid

VARIABLE move

vars ≜ ⟨global_registers, local_registers, current_count, current_move, pc,
        stack, move⟩
```

```
99  Init ≜  Global variables
100         ∧ global_registers = [id ∈ RegisterIdx ↦ Nil]
101         ∧ local_registers = [id ∈ RegisterIdx ↦ Nil]
102         ∧ current_count = InstrCount
103         ∧ current_move = Nil
104          Procedure perform_move
105         ∧ move = defaultInitValue
106         ∧ stack = ⟨⟩
107         ∧ pc = "Run"

109 PerformMove ≜  ∧ pc = "PerformMove"
110                ∧ IF move.dst[1] = "global" ∧ move.src[1] = "literal"
111                     THEN  ∧ global_registers′ = [global_registers EXCEPT ![move.dst[2]] = move.src[2]]
112                           ∧ UNCHANGED local_registers
113                     ELSE  ∧ IF move.dst[1] = "local" ∧ move.src[1] = "literal"
114                                 THEN  ∧ local_registers′ = [local_registers EXCEPT ![move.dst[2]] = move
115                                       ∧ UNCHANGED global_registers
116                                 ELSE  ∧ IF move.dst[1] = "global" ∧ move.src[1] = "global"
117                                             THEN  ∧ global_registers′ = [global_registers EXCEPT ![move.
118                                                   ∧ UNCHANGED local_registers
119                                             ELSE  ∧ IF move.dst[1] = "local" ∧ move.src[1] = "local"
120                                                         THEN  ∧ local_registers′ = [local_registers EXCEP
121                                                               ∧ UNCHANGED global_registers
122                                                         ELSE  ∧ IF move.dst[1] = "global" ∧ move.src[1]
123                                                                     THEN  ∧ global_registers′ = [global_r
124                                                                           ∧ UNCHANGED local_register
125                                                                     ELSE  ∧ IF move.dst[1] = "local" ∧
126                                                                                 THEN  ∧ local_registers′
127                                                                                 ELSE  ∧ TRUE
128                                                                                       ∧ UNCHANGED l
129                                                                     ∧ UNCHANGED global_registe
130                ∧ pc′ = Head(stack).pc
131                ∧ move′ = Head(stack).move
132                ∧ stack′ = Tail(stack)
133                ∧ UNCHANGED ⟨current_count, current_move⟩

135 perform_move ≜  PerformMove

137 Run ≜  ∧ pc = "Run"
138        ∧ IF current_count > 0
139             THEN  ∧ current_count′ = current_count − 1
140                   ∧ ∃ src ∈ Registers :
141                       ∃ dst ∈ Registers :
142                         ∃ use_lit ∈ {TRUE, FALSE} :
143                           ∃ lit ∈ Literals :
144                             ∧ Assert((
```

```
                                    ∨ current_move = Nil
                                    ∨ ∧ current_move.dst[1] = "global"
                                      ∧ current_move.src[1] = "literal"
                                      ∧ global_registers[current_move.dst[2]] = current_move.src[2]
                                    ∨ ∧ current_move.dst[1] = "local"
                                      ∧ current_move.src[1] = "literal"
                                      ∧ local_registers[current_move.dst[2]] = current_move.src[2]
                                    ∨ ∧ current_move.dst[1] = "global"
                                      ∧ current_move.src[1] = "local"
                                      ∧ global_registers[current_move.dst[2]] = local_registers[current_n
                                    ∨ ∧ current_move.dst[1] = "local"
                                      ∧ current_move.src[1] = "local"
                                      ∧ local_registers[current_move.dst[2]] = local_registers[current_mo
                                    ∨ ∧ current_move.dst[1] = "global"
                                      ∧ current_move.src[1] = "global"
                                      ∧ global_registers[current_move.dst[2]] = global_registers[current_
                                    ∨ ∧ current_move.dst[1] = "local"
                                      ∧ current_move.src[1] = "global"
                                      ∧ local_registers[current_move.dst[2]] = global_registers[current_n
                                     ) = TRUE,
                                   "Failure of assertion at line 66, column 9.")
                            ∧ IF use_lit
                                 THEN ∧ current_move' = [src ↦ ⟨"literal", lit⟩,  dst ↦ dst]
                                 ELSE ∧ current_move' = [src ↦ src,  dst ↦ dst]
                            ∧ ∧ move' = current_move'
                              ∧ stack' = ⟨[procedure ↦ "perform_move",
                                           pc        ↦ "Run",
                                           move      ↦ move]⟩
                                           ∘ stack
                            ∧ pc' = "PerformMove"
                 ELSE  ∧ pc' = "Done"
                       ∧ UNCHANGED ⟨current_count, current_move, stack, move⟩
        ∧ UNCHANGED ⟨global_registers, local_registers⟩

 Allow infinite stuttering to prevent deadlock on termination.
Terminating ≜ pc = "Done" ∧ UNCHANGED vars

Next ≜ perform_move ∨ Run
       ∨ Terminating

Spec ≜ Init ∧ □[Next]_vars

Termination ≜ ◇(pc = "Done")

 END TRANSLATION
```

```
192 └──────────────────────────────────────────────────────────────────────┘
    \ * Modification History
    \ * Last modified Sat Dec 19 14:27:42 CET 2020 by ostera
    \ * Created Sat Dec 19 11:40:24 CET 2020 by ostera
```