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.

```
──────────── MODULE LAM_ISA_Move ────────────
EXTENDS Naturals, Sequences, TLC

CONSTANTS InstrCount, RegisterCount, Nil, Literals

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

Registers ≜ RegisterKinds × RegisterIdx

 --algorithm LAM_ISA_Move
variables
  registers = [r ∈ Registers ↦ Nil],
  current_count = InstrCount,
  current_move = Nil
  ;

define

  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


end define ;

procedure perform_move(move)begin
PerformMove:
  if move.src[1] = "literal" then registers[move.dst] := move.src[2]
   else registers[move.dst] := registers[move.src]
  end if ;
  return;
end procedure ;
```

```
begin
  $Run$:
    while $current\_count > 0$ do
      $current\_count := current\_count - 1$ ;
      with $src \in Registers$,
           $dst \in Registers$,
           $use\_lit \in \{\text{TRUE}, \text{FALSE}\}$,
           $lit \in Literals$ do
        Lets assert that the last move we did was actually carried out!
        assert (
              $\lor current\_move = Nil$
              $\lor \land current\_move.src[1] = \text{"literal"}$
                 $\land registers[current\_move.dst] = current\_move.src[2]$
              $\lor registers[current\_move.dst] = registers[current\_move.src]$
              $) = \text{TRUE}$ ;
        if $use\_lit$ then
          $current\_move := [src \mapsto \langle \text{"literal"}, lit \rangle, \; dst \mapsto dst]$ ;
         else
          $current\_move := [src \mapsto src, \; dst \mapsto dst]$ ;
        end if ;
        call $perform\_move(current\_move)$ ;
      end with ;
    end while ;
end algorithm   ;
 BEGIN TRANSLATION ($chksum(pcal) =$ "$abda9f12$" $\land chksum(tla) =$ "$99d227dd$")
CONSTANT $defaultInitValue$
VARIABLES $registers$, $current\_count$, $current\_move$, $pc$, $stack$

 define statement
$AllRegistersAreValid \triangleq \lor current\_move = Nil$
                           $\lor \land current\_move.dst[1] \in RegisterKinds$
                              $\land current\_move.dst[2] \in RegisterIdx$
                              $\land \lor \land current\_move.src[1] \in RegisterKinds$
                                    $\land current\_move.src[2] \in RegisterIdx$
                                 $\lor \land current\_move.src[1] \in \{\text{"literal"}\}$
                                    $\land current\_move.src[2] \in Literals$

$TypeInvariant \triangleq \land current\_count \in Nat$
                   $\land AllRegistersAreValid$

VARIABLE $move$

$vars \triangleq \langle registers, current\_count, current\_move, pc, stack, move \rangle$

$Init \triangleq$ Global variables
          $\land registers = [r \in Registers \mapsto Nil]$
          ∧ current_count = InstrCount
          ∧ current_move = Nil
           Procedure perform_move
          ∧ move = defaultInitValue
          ∧ stack = ⟨⟩
          ∧ pc = "Run"

PerformMove ≜ ∧ pc = "PerformMove"
              ∧ IF move.src[1] = "literal"
                    THEN ∧ registers' = [registers EXCEPT ![move.dst] = move.src[2]]
                    ELSE ∧ registers' = [registers EXCEPT ![move.dst] = registers[move.src]]
              ∧ pc' = Head(stack).pc
              ∧ move' = Head(stack).move
              ∧ stack' = Tail(stack)
              ∧ UNCHANGED ⟨current_count, current_move⟩

perform_move ≜ PerformMove

Run ≜ ∧ pc = "Run"
      ∧ IF current_count > 0
            THEN ∧ current_count' = current_count − 1
                 ∧ ∃ src ∈ Registers :
                      ∃ dst ∈ Registers :
                        ∃ use_lit ∈ {TRUE, FALSE} :
                          ∃ lit ∈ Literals :
                            ∧ 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,
                                      "Failure of assertion at line 62, 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 registers

 Allow infinite stuttering to prevent deadlock on termination.
```

$Terminating \triangleq pc =$ "Done" $\land$ UNCHANGED $vars$

$Next \triangleq perform\_move \lor Run$
$\qquad\qquad \lor Terminating$

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

$Termination \triangleq \Diamond(pc =$ "Done"$)$

END TRANSLATION

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