MODULE *bank_account_assembly*

```
Joint bank account by husband and wife; Only assembly statements (not C)
  are assumed atomic. This version will assert an error when total ≠ 120,
  even though initially, account = 100, and
                      cash["husband"] = cash["wife"] = 10.
Note that if you remove the labels
    w0b, w0c, w1b, d0b, d0c, d1b, then there will be no assertion error.

EXTENDS Naturals, Sequences, TLC   Sequences required for "procedure" stmt
CONSTANT N   N is number of iterations. Assign to it in model overview.

--algorithm bank{
  variables account = 100, cash = [i ∈ {"husband", "wife"} ↦ 10],
                 iterations = [i ∈  {"husband", "wife"} ↦ N] ;
      Note that we need to define iterations["husband"] and iterations["wife"].
        We do _not_ want a single global variable, iterations, that is
        shared between "husband" and "wife".
      In model, replace defaultInitValue by value for iterations

    The procedures withdraw and deposit have been translated here
      to pseudo-assembly language
    Note that "register1" and "register2" were declared as local variables
      inside the processes for husband and wife.
  procedure withdraw( amount1 )
    variable register1, register2 ;
  {
    withdraw_start: register1 := amount1 ;           lw register1, (amount1)
     w0b:                 register2 := account − register1 ;   lw register2, (account) ; sub register2, register2, register1
     w0c:                 account := register2 ;         sw register2, (account)

     w1:              register2 := cash[self] + register1 ;   lw register2, (cash[self]) ; add register2, register2, register1
     w1b:                 cash[self] := register2 ;       sw register2, (cash[self])

     w2:              return ;
   }

  procedure deposit( amount1 )
    variable register1, register2 ;
  {
   deposit_start: register1 := amount1 ;              lw register1, (amount1)
     d0b:           register2 := account + register1 ;   lw register2, (account)
                                                   add register2, register2, register1
     d0c:           account := register2 ;              sw register2, (account)
    $d1$:        $register2 := cash[self] - register1$ ;
                                     lw $register2$, $(cash[self])$
                                     sub $register2$, $register2$, $register1$
    $d1b$:       $cash[self] := register2$ ;       sw $register2$, $(cash[self])$

    $d2$:        **return** ;
  }

process ( $spouse \in \{$"husband", "wife"$\}$ )
  **variable** $total$ ;
{ $start$: **while** ( $iterations[self] > 0$ ) {
      We hard-wire the max amount below, but this could have been a CONSTANT .
    $s1$: **with** ( $amount \in 1 \,..\, 2$ )
          **call** $withdraw(amount)$ ;
    $s2$: **with** ( $amount \in 1 \,..\, 2$ )
          **call** $deposit(amount)$ ;
    $s3$: $iterations[self] := iterations[self] - 1$ ;
        $total := account + cash[$"husband"$] + cash[$"wife"$]$ ;
     } ;
    **assert** $iterations[self] = 0$ ;

  $the\_end$: **if** ( $iterations[$"husband"$] = 0 \land iterations[$"wife"$] = 0$ ) {
        $total := account + cash[$"husband"$] + cash[$"wife"$]$ ;
        **print** $total$ ;
        **assert** $total = 120$ ;
     }
  }  end process block

}  \* end algorithm
```