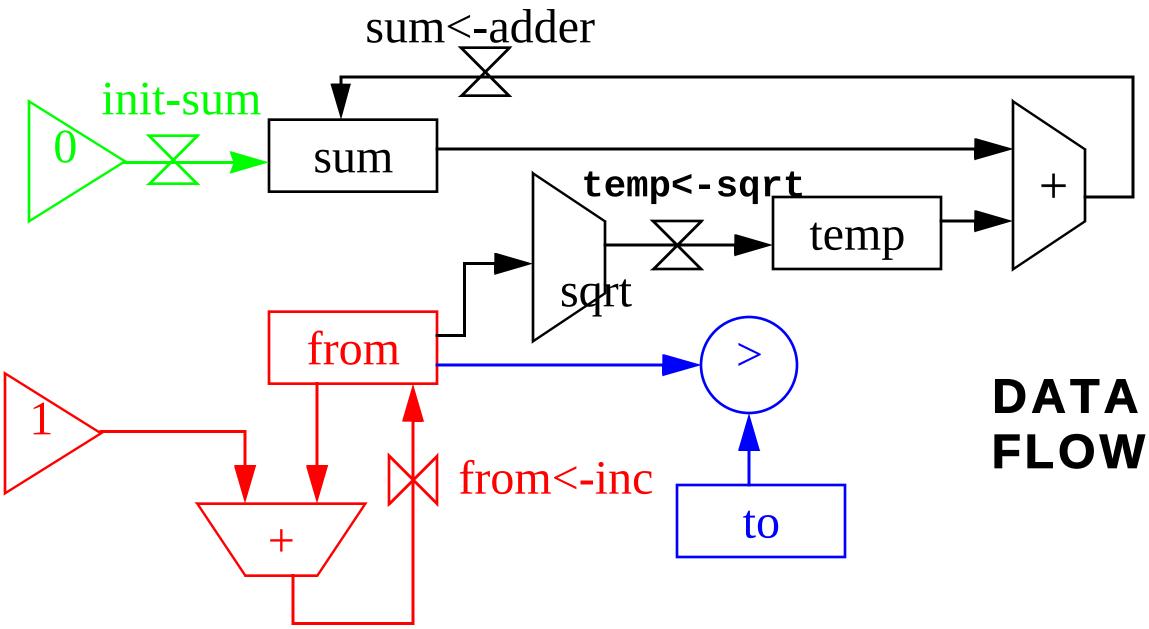
Sum-Roots Procedure

```
(define (sum-roots from to)
  (define (sum-iter sum from to)
    (if (> from to)
        sum
        (sum-iter (+ sum (sqrt from))
                  (+ 1 from)
                  to)))
  (sum-iter 0 from to))
```

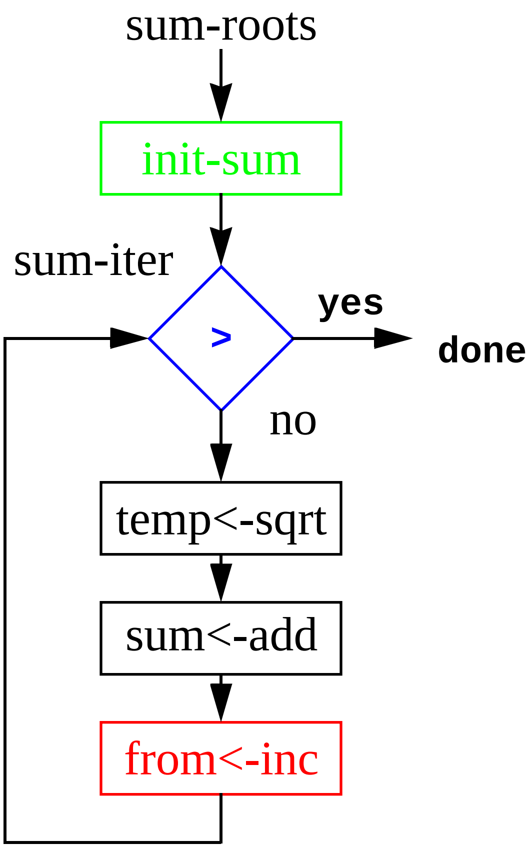
Register Machine for Sum-Roots

```
(define-machine sum-roots
 (registers sum from to temp)
 (operations + sqrt >)
 (controller
   sum-roots
     (assign sum (const 0))
   sum-iter
     (test (op >) (reg from) (reg to))
     (branch (label done))
     (assign temp (op sqrt) (reg from))
     (assign sum (op +) (reg sum) (reg temp))
     (assign from (op +) (const 1) (reg from))
     (goto (label sum-iter))
   done))
```

Sum-Roots Data Flow and Control Flow:



DATA FLOW



CONTROL FLOW

Register Machine Controller Language

```
(assign <reg-name1> (reg <reg-name2>))
(assign <reg-name> (const <constant-value>))
(assign <reg-name> (op <op-name>) <input1> <input2> ...)


(perform (op <op-name>) <input1> <input2> ...)

(test (op <op-name>) <input1> <input2> ...)
(branch (label <label-name>))
(goto (label <label-name>))
```

Notes:

```
<inputi> is either (const <constant-value>) or
                   (reg <reg-name>)
```

... thus no "nested" operations are allowed.

Register Machine Controller Language

```
(assign <reg-name1> (reg <reg-name2>))
(assign <reg-name> (const <constant-value>))
(assign <reg-name> (op <op-name>) <input1> <input2> ...)
(assign <reg-name> (label <label-name>))

(perform (op <op-name>) <input1> <input2> ...)

(test (op <op-name>) <input1> <input2> ...)
(branch (label <label-name>))
(goto (label <label-name>))
(goto (reg <reg-name>))
```

Notes:

```
<inputi> is either (const <constant-value>) or
                   (reg <reg-name>)
```

... thus no "nested" operations are allowed.

Register Machine Controller Language

```
(assign <reg-name1> (reg <reg-name2>))
(assign <reg-name> (const <constant-value>))
(assign <reg-name> (op <op-name>) <input1> <input2> ...)
(assign <reg-name> (label <label-name>))

(perform (op <op-name>) <input1> <input2> ...)

(test (op <op-name>) <input1> <input2> ...)
(branch (label <label-name>))
(goto (label <label-name>))
(goto (reg <reg-name>))

(save <reg-name>)
(restore <reg-name>)
```

Notes:

```
<inputi> is either (const <constant-value>) or
                   (reg <reg-name>)
```

... thus no "nested" operations are allowed.

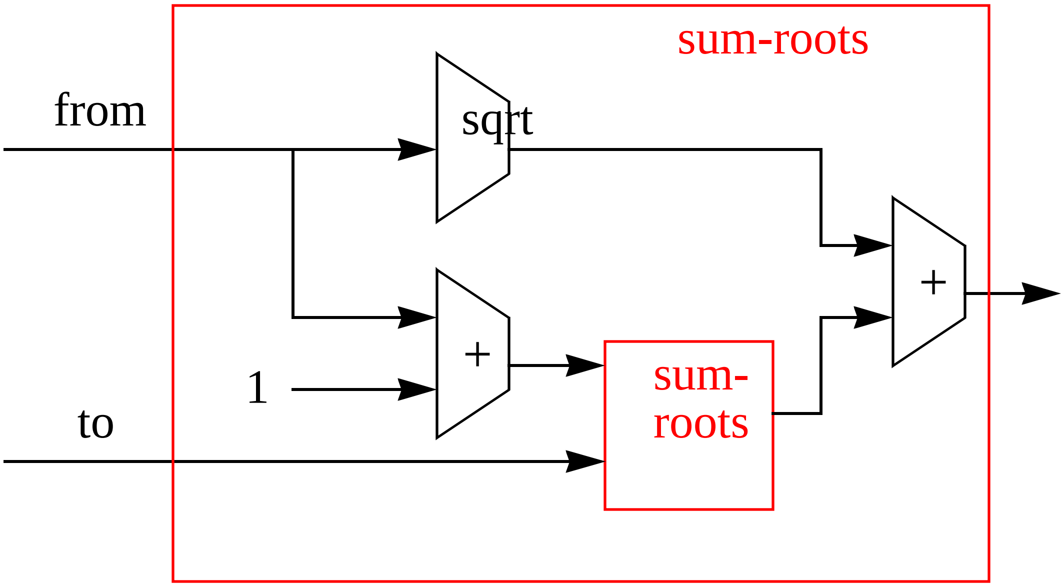
Subroutine Version of Sum-Roots

```
(controller

 ...
 ;; Contract: input registers from, to
 ;;           output in register sum
 ;; Returns to label in the continue register.
 sum-roots   ;; entry point
  (assign sum (const 0))
 sum-iter
  (test (op >) (reg from) (reg to))
  (branch (label done))
  (assign temp (op sqrt) (reg from))
  (assign sum (op +) (reg sum) (reg temp))
  (assign from (op +) (const 1) (reg from))
  (goto (label sum-iter))
 done
  (goto (reg continue))
 ...
 )
```

Recursive Sum-Roots

```
(define (sum-roots from to)
  (if (> from to)
      0                         ;; base case
      (+ (sqrt from)            ;; deferred operation
         (sum-roots (+ 1 from) to))))   ;; recursion
```



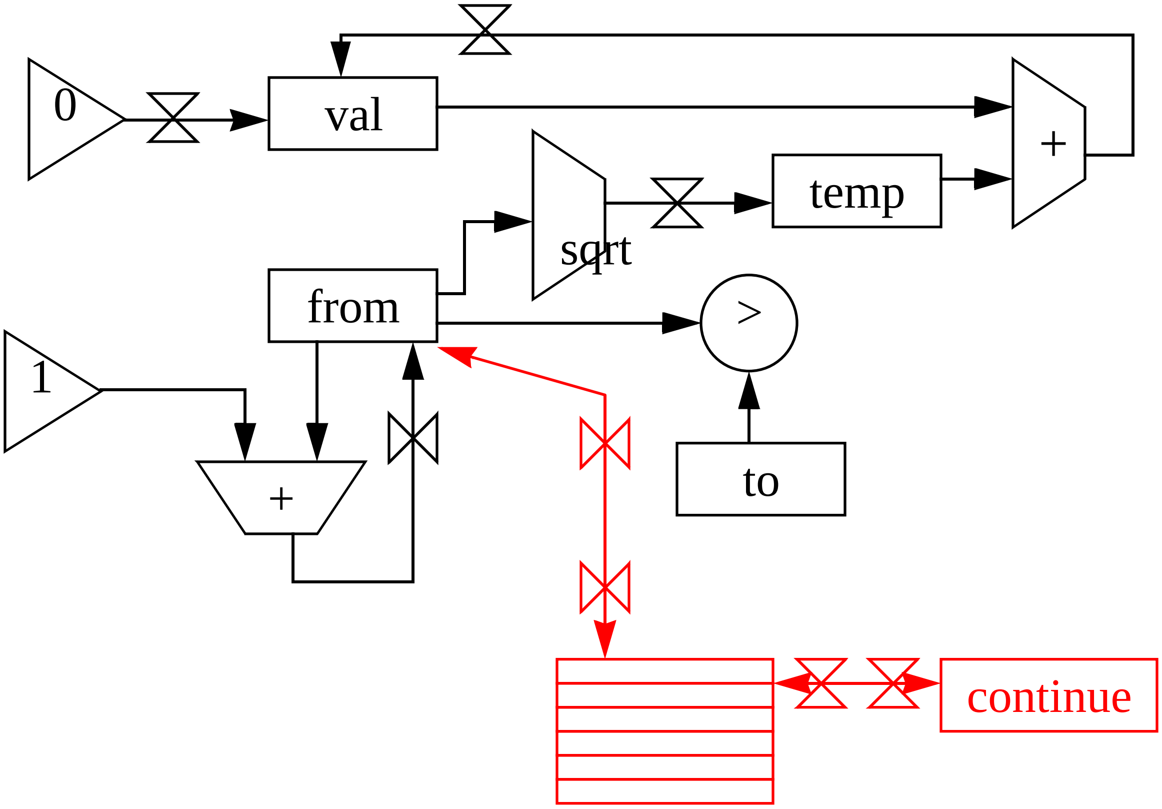
Recursive Sum-Roots (Failed Attempt)

```
(controller
 ;; On entry  -- continue holds return label
 ;;           -- registers from, to hold input values
 ;; On return -- register val holds answer
 sum-roots
  (test (op >) (reg from) (reg to))
  (branch (label base-case))
  ;; Need to recurse, so remember what we’ll need
  ;; for the deferred operation...
  (assign old-continue (reg continue))
  (assign old-from (reg from))
  (assign continue (label do-deferred-operations))
  (assign from (op +) (const 1) (reg from))
  (goto (label sum-roots))  ; recurse
 base-case
  (assign val (const 0))
  (goto (reg continue))
 do-deferred-operations
  (assign from (reg old-from))
  (assign temp (op sqrt) (reg from))
  (assign val (op +) (reg val) (reg temp))
  (assign continue (reg old-continue))
  (goto (reg continue))
)
```

Recursive Sum-Roots (With Stack)

```
(define-machine sum-roots
 (registers continue from to temp)
 (operations + sqrt >)
 (controller
  ;; On entry  -- continue holds return label
  ;;          -- registers from, to hold input values
  ;; On return -- register val holds answer
  sum-roots
    (test (op >) (reg from) (reg to))
    (branch (label base-case))
    ;; Need to recurse, so remember what we’ll need
    ;; for the deferred operation...
    (save from)
    (save continue)
    (assign continue (label do-deferred-operations))
    (assign from (op +) (const 1) (reg from))
    (goto (label sum-roots))  ; recurse
  base-case
    (assign val (const 0))
    (goto (reg continue))
  do-deferred-operations
    (restore continue)  ;; restore in reverse order!
    (restore from)
    (assign temp (op sqrt) (reg from))
    (assign val (op +) (reg val) (reg temp))
    (goto (reg continue))))
```

Recursive Sum-Roots Register Machine


0
val
+
temp
sqrt
from
>
1
+
to
continue