```
#|

 Copyright (C) 1994 by Yuan Yu.  All Rights Reserved.

 This script is hereby placed in the public domain, and therefore unlimited
 editing and redistribution is permitted.

 NO WARRANTY

 Yuan Yu PROVIDES ABSOLUTELY NO WARRANTY.  THE EVENT SCRIPT IS PROVIDED "AS IS"
 WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT
 LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 PARTICULAR PURPOSE.  THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE
 SCRIPT IS WITH YOU.  SHOULD THE SCRIPT PROVE DEFECTIVE, YOU ASSUME THE COST OF
 ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

 IN NO EVENT WILL Yuan Yu BE LIABLE TO YOU FOR ANY DAMAGES, ANY LOST PROFITS,
 LOST MONIES, OR OTHER SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT
 OF THE USE OR INABILITY TO USE THIS SCRIPT (INCLUDING BUT NOT LIMITED TO LOSS
 OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY THIRD
 PARTIES), EVEN IF YOU HAVE ADVISED US OF THE POSSIBILITY OF SUCH DAMAGES, OR
 FOR ANY CLAIM BY ANY OTHER PARTY.

|#
```

EVENT: Start with the library "mc20-2" using the compiled version.

```
;            Proof of the Correctness of a ZERO Program
#|
```

This program zeros an integer array.

```
/* zeros an array */
zero(a, n)
int a[], n;
{
  int i;
  for (i = 0; i < n; ++i)
    a[i] = 0;
}
```

Here, in symbolic form, is the above program in MC68k instructions. The machine code is produced by "gcc -O".

```
a6@8,  the first argument which is the starting address of the array A.
a6@12, the second argument which is the number of the elements we are zeroing.
a6@-4, the local variable i which serves as a counter.

0x2328 <zero>:         linkw a6,#0
0x232c <zero+4>: moveal a6@(8),a0
0x2330 <zero+8>: movel a6@(12),d1
0x2334 <zero+12>: clrl d0
0x2336 <zero+14>: cmpl d0,d1
0x2338 <zero+16>: ble 0x2344 <zero+28>
0x233a <zero+18>: clrl 0(a0)[d0.l*4]
0x233e <zero+22>: addql #1,d0
0x2340 <zero+24>: cmpl d0,d1
0x2342 <zero+26>: bgt 0x233a <zero+18>
0x2344 <zero+28>: unlk a6
0x2346 <zero+30>: rts

The machine code of the above program is:

<zero>:      0x4e56  0x0000  0x206e  0x0008  0x222e  0x000c  0x4280  0xb280
<zero+16>:   0x6f0a  0x42b0  0x0c00  0x5280  0xb280  0x6ef6  0x4e5e  0x4e75

'(78      86      0       0       32      110     0       8
  34      46      0       12      66      128     178     128
  111     10      66      176     12      0       82      128
  178     128     110     246     78      94      78      117)
|#

; machine code for ZERO.
```

DEFINITION:
ZERO-CODE
= `'(78 86 0 0 32 110 0 8 34 46 0 12 66 128 178 128 111 10 66 176 12 0 82 128 178 128 110 246 78 94 78 117)`

DEFINITION:
clr-lst $(i,\ n,\ lst)$
= **if** $i < n$ **then** clr-lst $(1 + i,\ n,$ put-nth $(0,\ i,\ lst))$
else lst **endif**

DEFINITION:
zero1-t $(i,\ n)$
= **if** $i < n$ **then** splus $(4,$ zero1-t $(1 + i,\ n))$
else 4 **endif**

DEFINITION:
zero-t (n)
= **if** $n \simeq$ `0` **then** `8`
else splus (`8`, zero1-t (`1`, n)) **endif**

DEFINITION:
zero-induct (s, i, n, *lst*)
= **if** $i < n$ **then** zero-induct (stepn (s, `4`), $1 + i$, n, put-nth (`0`, i, *lst*))
else t endif

DEFINITION:
zero-statep (s, a, n, *lst*)
= ((mc-status (s) = `'running`)
∧ evenp (mc-pc (s))
∧ rom-addrp (mc-pc (s), mc-mem (s), `32`)
∧ mcode-addrp (mc-pc (s), mc-mem (s), ZERO-CODE)
∧ ram-addrp (sub (`32`, `4`, read-sp (s)), mc-mem (s), `16`)
∧ mem-lst (`4`, a, mc-mem (s), n, *lst*)
∧ ram-addrp (a, mc-mem (s), `4` $*$ n)
∧ disjoint (a, `4` $*$ n, sub (`32`, `4`, read-sp (s)), `16`)
∧ (a = read-mem (add (`32`, read-sp (s), `4`), mc-mem (s), `4`))
∧ (n = iread-mem (add (`32`, read-sp (s), `8`), mc-mem (s), `4`))
∧ uint-rangep (`4` $*$ n, `32`)
∧ ($n \in \mathbf{N}$))

DEFINITION:
zero-s0p (s, a, i, n, *lst*)
= ((mc-status (s) = `'running`)
∧ evenp (mc-pc (s))
∧ rom-addrp (sub (`32`, `24`, mc-pc (s)), mc-mem (s), `32`)
∧ mcode-addrp (sub (`32`, `24`, mc-pc (s)), mc-mem (s), ZERO-CODE)
∧ ram-addrp (read-an (`32`, `6`, s), mc-mem (s), `16`)
∧ mem-lst (`4`, a, mc-mem (s), n, *lst*)
∧ ram-addrp (a, mc-mem (s), `4` $*$ n)
∧ disjoint (a, `4` $*$ n, read-an (`32`, `6`, s), `16`)
∧ (a = read-an (`32`, `0`, s))
∧ (i = nat-to-int (read-dn (`32`, `0`, s), `32`))
∧ (n = nat-to-int (read-dn (`32`, `1`, s), `32`))
∧ uint-rangep (`4` $*$ n, `32`)
∧ ($i \in \mathbf{N}$)
∧ ($n \in \mathbf{N}$)
∧ ($i \leq n$))

```
; from s to exit: s --> sn.
```

THEOREM: zero-s-sn
(zero-statep (*s*, *a*, *n*, *lst*) ∧ (*n* ≃ 0))
→ ((mc-status (stepn (*s*, 8)) = 'running)
∧ (mc-pc (stepn (*s*, 8)) = rts-addr (*s*))
∧ mem-lst (4, *a*, mc-mem (stepn (*s*, 8)), *n*, *lst*)
∧ (read-rn (32, 14, mc-rfile (stepn (*s*, 8)))
= read-rn (32, 14, mc-rfile (*s*)))
∧ (read-rn (32, 15, mc-rfile (stepn (*s*, 8)))
= add (32, read-rn (32, 15, mc-rfile (*s*)), 4)))

THEOREM: zero-s-sn-rfile
(zero-statep (*s*, *a*, *n*, *lst*) ∧ d2-7a2-5p (*rn*) ∧ (*n* ≃ 0))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 8)))
= read-rn (*oplen*, *rn*, mc-rfile (*s*)))

THEOREM: zero-s-sn-mem
(zero-statep (*s*, *a*, *n*, *lst*)
∧ disjoint (sub (32, 4, read-sp (*s*)), 16, *x*, *k*)
∧ (*n* ≃ 0))
→ (read-mem (*x*, mc-mem (stepn (*s*, 8)), *k*) = read-mem (*x*, mc-mem (*s*), *k*))

```
; from s to s0: s --> s0.
```

THEOREM: zero-s-s0
(zero-statep (*s*, *a*, *n*, *lst*) ∧ (*n* ≄ 0))
→ zero-s0p (stepn (*s*, 8), *a*, 1, *n*, put-nth (0, 0, *lst*))

THEOREM: zero-s-s0-other
(zero-statep (*s*, *a*, *n*, *lst*) ∧ (*n* ≄ 0))
→ ((linked-rts-addr (stepn (*s*, 8)) = rts-addr (*s*))
∧ (linked-a6 (stepn (*s*, 8)) = read-an (32, 6, *s*))
∧ (read-rn (32, 14, mc-rfile (stepn (*s*, 8)))
= sub (32, 4, read-sp (*s*))))

THEOREM: zero-s-s0-rfile
(zero-statep (*s*, *a*, *n*, *lst*) ∧ d2-7a2-5p (*rn*) ∧ (*n* ≄ 0))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 8)))
= read-rn (*oplen*, *rn*, mc-rfile (*s*)))

THEOREM: zero-s-s0-mem
(zero-statep (*s*, *a*, *n*, *lst*)
∧ disjoint (sub (32, 4, read-sp (*s*)), 16, *x*, *k*)
∧ disjoint (*a*, 4 ∗ *n*, *x*, *k*)
∧ (*n* ≄ 0))
→ (read-mem (*x*, mc-mem (stepn (*s*, 8)), *k*) = read-mem (*x*, mc-mem (*s*), *k*))

```
; from s0 to exit:  s0 --> sn. By induction.
; base case: s0 --> sn, if i >= n.
```

THEOREM: zero-s0-sn-base
(zero-s0p (*s*, *a*, *i*, *n*, *lst*) ∧ (*i* ≮ *n*))
→ ((mc-status (stepn (*s*, 4)) = 'running)
∧ (mc-pc (stepn (*s*, 4)) = linked-rts-addr (*s*))
∧ mem-lst (4, *a*, mc-mem (stepn (*s*, 4)), *n*, *lst*)
∧ (read-rn (32, 14, mc-rfile (stepn (*s*, 4))) = linked-a6 (*s*))
∧ (read-rn (32, 15, mc-rfile (stepn (*s*, 4)))
= add (32, read-an (32, 6, *s*), 8)))

THEOREM: zero-s0-sn-mem-base
(zero-s0p (*s*, *a*, *i*, *n*, *lst*) ∧ (*i* ≮ *n*))
→ (read-mem (*x*, mc-mem (stepn (*s*, 4)), *k*) = read-mem (*x*, mc-mem (*s*), *k*))

THEOREM: zero-s0-sn-rfile-base
(zero-s0p (*s*, *a*, *i*, *n*, *lst*) ∧ d2-7a2-5p (*rn*) ∧ (*i* ≮ *n*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 4)))
= read-rn (*oplen*, *rn*, mc-rfile (*s*)))

```
; induction case: s0 --> s0, if i < n.
```

THEOREM: zero-s0-s0
(zero-s0p (*s*, *a*, *i*, *n*, *lst*) ∧ (*i* < *n*))
→ (zero-s0p (stepn (*s*, 4), *a*, 1 + *i*, *n*, put-nth (0, *i*, *lst*))
∧ (read-rn (*oplen*, 14, mc-rfile (stepn (*s*, 4)))
= read-rn (*oplen*, 14, mc-rfile (*s*)))
∧ (linked-a6 (stepn (*s*, 4)) = linked-a6 (*s*))
∧ (linked-rts-addr (stepn (*s*, 4)) = linked-rts-addr (*s*)))

THEOREM: zero-s0-s0-mem
(zero-s0p (*s*, *a*, *i*, *n*, *lst*) ∧ disjoint (*a*, 4 ∗ *n*, *x*, *k*) ∧ (*i* < *n*))
→ (read-mem (*x*, mc-mem (stepn (*s*, 4)), *k*) = read-mem (*x*, mc-mem (*s*), *k*))

THEOREM: zero-s0-s0-rfile
(zero-s0p (*s*, *a*, *i*, *n*, *lst*) ∧ d2-7a2-5p (*rn*) ∧ (*i* < *n*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 4)))
= read-rn (*oplen*, *rn*, mc-rfile (*s*)))

EVENT: Disable zero-statep.

EVENT: Disable zero-s0p.

```
; put together: s0 --> sn.
```

THEOREM: zero-s0-sn
zero-s0p (*s*, *a*, *i*, *n*, *lst*)
→ ((mc-status (stepn (*s*, zero1-t (*i*, *n*))) = 'running)
∧ (mc-pc (stepn (*s*, zero1-t (*i*, *n*))) = linked-rts-addr (*s*))
∧ mem-lst (4, *a*, mc-mem (stepn (*s*, zero1-t (*i*, *n*))), *n*, clr-lst (*i*, *n*, *lst*))
∧ (read-rn (32, 14, mc-rfile (stepn (*s*, zero1-t (*i*, *n*))))
= linked-a6 (*s*))
∧ (read-rn (32, 15, mc-rfile (stepn (*s*, zero1-t (*i*, *n*))))
= add (32, read-an (32, 6, *s*), 8)))

THEOREM: zero-s0-sn-mem
(zero-s0p (*s*, *a*, *i*, *n*, *lst*)
∧ disjoint (read-an (32, 6, *s*), 16, *x*, *k*)
∧ disjoint (*a*, 4 ∗ *n*, *x*, *k*))
→ (read-mem (*x*, mc-mem (stepn (*s*, zero1-t (*i*, *n*))), *k*)
= read-mem (*x*, mc-mem (*s*), *k*))

THEOREM: zero-s0-sn-rfile
(zero-s0p (*s*, *a*, *i*, *n*, *lst*) ∧ d2-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, zero1-t (*i*, *n*))))
= read-rn (*oplen*, *rn*, mc-rfile (*s*)))

; the correctness of the program ZERO.

THEOREM: zero-correctness
zero-statep (*s*, *a*, *n*, *lst*)
→ ((mc-status (stepn (*s*, zero-t (*n*))) = 'running)
∧ (mc-pc (stepn (*s*, zero-t (*n*))) = rts-addr (*s*))
∧ mem-lst (4, *a*, mc-mem (stepn (*s*, zero-t (*n*))), *n*, clr-lst (0, *n*, *lst*))
∧ (read-an (32, 6, stepn (*s*, zero-t (*n*))) = read-an (32, 6, *s*))
∧ (read-an (32, 7, stepn (*s*, zero-t (*n*)))
= add (32, read-an (32, 7, *s*), 4)))

THEOREM: zero-mem
(zero-statep (*s*, *a*, *n*, *lst*)
∧ disjoint (sub (32, 4, read-sp (*s*)), 16, *x*, *k*)
∧ disjoint (*a*, 4 ∗ *n*, *x*, *k*))
→ (read-mem (*x*, mc-mem (stepn (*s*, zero-t (*n*))), *k*) = read-mem (*x*, mc-mem (*s*), *k*))

THEOREM: zero-rfile
(zero-statep (*s*, *a*, *n*, *lst*) ∧ d2-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, zero-t (*n*))))
= read-rn (*oplen*, *rn*, mc-rfile (*s*)))

EVENT: Disable zero-t.

```
; we next need to prove that clr-lst does clear the list.
```

THEOREM: zero-lst-la1
$(j < i) \rightarrow (\text{get-nth}\,(j,\ \text{clr-lst}\,(i,\ n,\ lst)) = \text{get-nth}\,(j,\ lst))$

THEOREM: zero-lst-la2
get-nth $(i,$ clr-lst $(i,\ n,\ lst))$
$=$ **if** $i < n$ **then** 0
else get-nth $(i,\ lst)$ **endif**

```
; clr-lst does clear the list.
```

THEOREM: clr-lst-clear
$((i \leq j) \wedge (j < n)) \rightarrow (\text{get-nth}\,(j,\ \text{clr-lst}\,(i,\ n,\ lst)) = 0)$

THEOREM: clr-lst-clear-0
$(j < n) \rightarrow (\text{get-nth}\,(j,\ \text{clr-lst}\,(0,\ n,\ lst)) = 0)$

Index

add, 3–6

clr-lst, 2, 6, 7
clr-lst-clear, 7
clr-lst-clear-0, 7

d2-7a2-5p, 4–6
disjoint, 3–6

evenp, 3

get-nth, 7

iread-mem, 3

linked-a6, 4–6
linked-rts-addr, 4–6

mc-mem, 3–6
mc-pc, 3–6
mc-rfile, 4–6
mc-status, 3–6
mcode-addrp, 3
mem-lst, 3–6

nat-to-int, 3

put-nth, 2–5

ram-addrp, 3
read-an, 3–6
read-dn, 3
read-mem, 3–6
read-rn, 4–6
read-sp, 3, 4, 6
rom-addrp, 3
rts-addr, 4, 6

splus, 2, 3
stepn, 3–6
sub, 3, 4, 6

uint-rangep, 3

zero-code, 2, 3
zero-correctness, 6
zero-induct, 3
zero-lst-la1, 7
zero-lst-la2, 7
zero-mem, 6
zero-rfile, 6
zero-s-s0, 4
zero-s-s0-mem, 4
zero-s-s0-other, 4
zero-s-s0-rfile, 4
zero-s-sn, 4
zero-s-sn-mem, 4
zero-s-sn-rfile, 4
zero-s0-s0, 5
zero-s0-s0-mem, 5
zero-s0-s0-rfile, 5
zero-s0-sn, 6
zero-s0-sn-base, 5
zero-s0-sn-mem, 6
zero-s0-sn-mem-base, 5
zero-s0-sn-rfile, 6
zero-s0-sn-rfile-base, 5
zero-s0p, 3–6
zero-statep, 3, 4, 6
zero-t, 3, 6
zero1-t, 2, 3, 6