```
#|

 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.

|#

;            Proof of the Correctness of a GCD Program
```

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

```
#|

The following C program computes the greatest common divisor of two
integers a and b. We, here, investigate the machine code of this
program generated by a widely used C compiler, and verify the
correctness of the code.

/* computes the greatest common divisor by Euclid's algorithm */
gcd(int a, int b)
{
  while (a != 0){
    if (b == 0) return (a);
    if (a > b)
      a = a % b;
    else b = b % a;
```

```
  };
  return (b);
}
```

Here is the MC68020 assembly code of the above GCD program. The code is generated by "gcc -O".

```
0x22c6 <gcd>:          linkw a6,#0
0x22ca <gcd+4>:        moveml d2-d3,sp@-
0x22ce <gcd+8>:        movel a6@(8),d2
0x22d2 <gcd+12>:       movel a6@(12),d3
0x22d6 <gcd+16>:       tstl d2
0x22d8 <gcd+18>:       beq 0x22f6 <gcd+48>
0x22da <gcd+20>:       tstl d3
0x22dc <gcd+22>:       bne 0x22e2 <gcd+28>
0x22de <gcd+24>:       movel d2,d0
0x22e0 <gcd+26>:       bra 0x22f8 <gcd+50>
0x22e2 <gcd+28>:       cmpl d2,d3
0x22e4 <gcd+30>:       bge 0x22ee <gcd+40>
0x22e6 <gcd+32>:       divsll d3,d0,d2
0x22ea <gcd+36>:       movel d0,d2
0x22ec <gcd+38>:       bra 0x22d6 <gcd+16>
0x22ee <gcd+40>:       divsll d2,d0,d3
0x22f2 <gcd+44>:       movel d0,d3
0x22f4 <gcd+46>:       bra 0x22d6 <gcd+16>
0x22f6 <gcd+48>:       movel d3,d0
0x22f8 <gcd+50>:       moveml a6@(-8),d2-d3
0x22fe <gcd+56>:       unlk a6
0x2300 <gcd+58>:       rts
```

The machine code of the above program is:

```
<gcd>:       0x4e56  0x0000  0x48e7  0x3000  0x242e  0x0008  0x262e  0x000c
<gcd+16>:    0x4a82  0x671c  0x4a83  0x6604  0x2002  0x6016  0xb682  0x6c08
<gcd+32>:    0x4c43  0x2800  0x2400  0x60e8  0x4c42  0x3800  0x2600  0x60e0
<gcd+48>:    0x2003  0x4cee  0x000c  0xfff8  0x4e5e  0x4e75

'(78      86      0       0       72      231     48      0
  36      46      0       8       38      46      0       12
  74      130     103     28      74      131     102     4
  32      2       96      22      182     130     108     8
  76      67      40      0       36      0       96      232
  76      66      56      0       38      0       96      224
  32      3       76      238     0       12      255     248
```

```
  78      94      78      117)
|#

; now we start the correctness proof of this GCD program, defined by
; (gcd-code).
```

DEFINITION:
GCD-CODE

```
=  '(78 86 0 0 72 231 48 0 36 46 0 8 38 46 0 12 74 130
     103 28 74 131 102 4 32 2 96 22 182 130 108 8 76 67
     40 0 36 0 96 232 76 66 56 0 38 0 96 224 32 3 76 238
     0 12 255 248 78 94 78 117)
```

CONSERVATIVE AXIOM: gcd-load

gcd-loadp (s)

$$
\begin{aligned}
= \quad & (\text{evenp}\,(\text{GCD-ADDR}) \\
& \quad \land \quad (\text{GCD-ADDR} \in \mathbf{N}) \\
& \quad \land \quad \text{nat-rangep}\,(\text{GCD-ADDR},\ 32) \\
& \quad \land \quad \text{rom-addrp}\,(\text{GCD-ADDR},\ \text{mc-mem}\,(s),\ 60) \\
& \quad \land \quad \text{mcode-addrp}\,(\text{GCD-ADDR},\ \text{mc-mem}\,(s),\ \text{GCD-CODE}))
\end{aligned}
$$

Simultaneously, we introduce the new function symbols *gcd-loadp* and *gcd-addr*.

THEOREM: stepn-gcd-loadp

gcd-loadp (stepn $(s,\ n)$) = gcd-loadp (s)

```
; the functional description of the program (gcd-code).
```

DEFINITION:

gcd $(a,\ b)$

= **if** $a \simeq 0$ **then** fix (b)
elseif $b \simeq 0$ **then** a
elseif $b < a$ **then** gcd $(a$ **mod** $b,\ b)$
else gcd $(a,\ b$ **mod** $a)$ **endif**

```
; the clock function.
```

DEFINITION:

gcd-t1 $(a,\ b)$

= **if** $a \simeq 0$ **then** 6
elseif $b \simeq 0$ **then** 9
elseif $b < a$ **then** splus (9, gcd-t1 $(a$ **mod** $b,\ b)$)
else splus (9, gcd-t1 $(a,\ b$ **mod** $a)$) **endif**

DEFINITION: gcd-t $(a,\ b)$ = splus (4, gcd-t1 $(a,\ b)$)

```
; an induction hint.
```

DEFINITION:
gcd-induct $(s,\ a,\ b)$
$=$ **if** $(a \simeq 0) \lor (b \simeq 0)$ **then t**
elseif $b < a$ **then** gcd-induct (stepn $(s,\ 9),\ a$ **mod** $b,\ b)$
else gcd-induct (stepn $(s,\ 9),\ a,\ b$ **mod** $a)$ **endif**

```
; the initial state.
```

DEFINITION:
gcd-statep $(s,\ a,\ b)$
$=$ ((mc-status (s) = 'running)
$\land$ evenp (mc-pc (s))
$\land$ rom-addrp (mc-pc (s), mc-mem (s), 60)
$\land$ mcode-addrp (mc-pc (s), mc-mem (s), GCD-CODE)
$\land$ ram-addrp (sub (32, 12, read-sp (s)), mc-mem (s), 24)
$\land$ $(a =$ iread-mem (add (32, read-sp (s), 4), mc-mem (s), 4))
$\land$ $(b =$ iread-mem (add (32, read-sp (s), 8), mc-mem (s), 4))
$\land$ $(a \in \mathbf{N})$
$\land$ $(b \in \mathbf{N})$)

```
; an intermediate state.
```

DEFINITION:
gcd-s0p $(s,\ a,\ b)$
$=$ ((mc-status (s) = 'running)
$\land$ evenp (mc-pc (s))
$\land$ rom-addrp (sub (32, 16, mc-pc (s)), mc-mem (s), 60)
$\land$ mcode-addrp (sub (32, 16, mc-pc (s)), mc-mem (s), GCD-CODE)
$\land$ ram-addrp (sub (32, 8, read-an (32, 6, s)), mc-mem (s), 24)
$\land$ $(a =$ iread-dn (32, 2, s))
$\land$ $(b =$ iread-dn (32, 3, s))
$\land$ $(a \in \mathbf{N})$
$\land$ $(b \in \mathbf{N})$)

```
; s --> s0.
```

THEOREM: gcd-s-s0
gcd-statep $(s,\ a,\ b)$
$\rightarrow$ (gcd-s0p (stepn $(s,\ 4),\ a,\ b)$
$\land$ (linked-rts-addr (stepn $(s,\ 4)$) = rts-addr (s))
$\land$ (linked-a6 (stepn $(s,\ 4)$) = read-an (32, 6, s))
$\land$ (read-rn (32, 14, mc-rfile (stepn $(s,\ 4)$))
$=$ sub (32, 4, read-sp (s)))
$\land$ (movem-saved (stepn $(s,\ 4)$, 4, 8, 2)
$=$ readm-rn (32, '(2 3), mc-rfile (s))))

THEOREM: gcd-s-s0-rfile
$(\text{gcd-statep}(s, a, b) \wedge \text{d4-7a2-5p}(rn))$
$\rightarrow \quad (\text{read-rn}(oplen, rn, \text{mc-rfile}(\text{stepn}(s, 4)))$
$\quad = \quad \text{read-rn}(oplen, rn, \text{mc-rfile}(s)))$

THEOREM: gcd-s-s0-mem
$(\text{gcd-statep}(s, a, b) \wedge \text{disjoint}(x, l, \text{sub}(32, 12, \text{read-sp}(s)), 24))$
$\rightarrow \quad (\text{read-mem}(x, \text{mc-mem}(\text{stepn}(s, 4)), l) = \text{read-mem}(x, \text{mc-mem}(s), l))$

```
; s0 --> exit.
; base case: s0 --> exit.
```

THEOREM: gcd-s0-sn-base-1
$(\text{gcd-s0p}(s, a, b) \wedge (a \simeq 0))$
$\rightarrow \quad ((\text{mc-status}(\text{stepn}(s, 6)) = \texttt{'running})$
$\quad \wedge \quad (\text{mc-pc}(\text{stepn}(s, 6)) = \text{linked-rts-addr}(s))$
$\quad \wedge \quad (\text{iread-dn}(32, 0, \text{stepn}(s, 6)) = \text{fix}(b))$
$\quad \wedge \quad (\text{read-rn}(32, 14, \text{mc-rfile}(\text{stepn}(s, 6))) = \text{linked-a6}(s))$
$\quad \wedge \quad (\text{read-rn}(32, 15, \text{mc-rfile}(\text{stepn}(s, 6)))$
$\qquad = \quad \text{add}(32, \text{read-an}(32, 6, s), 8))$
$\quad \wedge \quad (\text{read-mem}(x, \text{mc-mem}(\text{stepn}(s, 6)), l)$
$\qquad = \quad \text{read-mem}(x, \text{mc-mem}(s), l)))$

THEOREM: gcd-s0-sn-base-rfile-1
$(\text{gcd-s0p}(s, a, b) \wedge (a \simeq 0) \wedge \text{d2-7a2-5p}(rn) \wedge (oplen \leq 32))$
$\rightarrow \quad (\text{read-rn}(oplen, rn, \text{mc-rfile}(\text{stepn}(s, 6)))$
$\quad = \quad \textbf{if } \text{d4-7a2-5p}(rn) \textbf{ then } \text{read-rn}(oplen, rn, \text{mc-rfile}(s))$
$\qquad \textbf{else } \text{get-vlst}(oplen, 0, rn, \texttt{'(2 3)}, \text{movem-saved}(s, 4, 8, 2)) \textbf{ endif})$

THEOREM: gcd-s0-sn-base-2
$(\text{gcd-s0p}(s, a, b) \wedge (a \not\simeq 0) \wedge (b \simeq 0))$
$\rightarrow \quad ((\text{mc-status}(\text{stepn}(s, 9)) = \texttt{'running})$
$\quad \wedge \quad (\text{mc-pc}(\text{stepn}(s, 9)) = \text{linked-rts-addr}(s))$
$\quad \wedge \quad (\text{iread-dn}(32, 0, \text{stepn}(s, 9)) = \text{fix}(a))$
$\quad \wedge \quad (\text{read-rn}(32, 14, \text{mc-rfile}(\text{stepn}(s, 9))) = \text{linked-a6}(s))$
$\quad \wedge \quad (\text{read-rn}(32, 15, \text{mc-rfile}(\text{stepn}(s, 9)))$
$\qquad = \quad \text{add}(32, \text{read-an}(32, 6, s), 8))$
$\quad \wedge \quad (\text{read-mem}(x, \text{mc-mem}(\text{stepn}(s, 9)), l)$
$\qquad = \quad \text{read-mem}(x, \text{mc-mem}(s), l)))$

THEOREM: gcd-s0-sn-base-rfile-2
$(\text{gcd-s0p}(s, a, b)$
$\wedge \quad (a \not\simeq 0)$
$\wedge \quad (b \simeq 0)$
$\wedge \quad \text{d2-7a2-5p}(rn)$

$\wedge \quad (oplen \leq \texttt{32}))$

$\rightarrow \quad (\text{read-rn}\,(oplen,\ rn,\ \text{mc-rfile}\,(\text{stepn}\,(s,\ \texttt{9})))$

$\quad\quad = \quad \textbf{if}\ \text{d4-7a2-5p}\,(rn)\ \ \textbf{then}\ \text{read-rn}\,(oplen,\ rn,\ \text{mc-rfile}\,(s))$

$\quad\quad\quad\quad \textbf{else}\ \text{get-vlst}\,(oplen,\ \texttt{0},\ rn,\ \texttt{'(2 3)},\ \text{movem-saved}\,(s,\ \texttt{4},\ \texttt{8},\ \texttt{2}))\ \textbf{endif})$

```
; induction case:  s0 --> s0.
```

EVENT: Enable integerp.

EVENT: Enable iremainder.

THEOREM: gcd-s0-s0-1

$(\text{gcd-s0p}\,(s,\ a,\ b) \wedge (a \not\simeq \texttt{0}) \wedge (b \not\simeq \texttt{0}) \wedge (b < a))$

$\rightarrow \quad (\text{gcd-s0p}\,(\text{stepn}\,(s,\ \texttt{9}),\ a\ \textbf{mod}\ b,\ b)$

$\quad\quad \wedge \quad (\text{read-rn}\,(oplen,\ \texttt{14},\ \text{mc-rfile}\,(\text{stepn}\,(s,\ \texttt{9})))$

$\quad\quad\quad\quad = \quad \text{read-rn}\,(oplen,\ \texttt{14},\ \text{mc-rfile}\,(s)))$

$\quad\quad \wedge \quad (\text{linked-a6}\,(\text{stepn}\,(s,\ \texttt{9})) = \text{linked-a6}\,(s))$

$\quad\quad \wedge \quad (\text{linked-rts-addr}\,(\text{stepn}\,(s,\ \texttt{9})) = \text{linked-rts-addr}\,(s))$

$\quad\quad \wedge \quad (\text{movem-saved}\,(\text{stepn}\,(s,\ \texttt{9}),\ \texttt{4},\ \texttt{8},\ \texttt{2}) = \text{movem-saved}\,(s,\ \texttt{4},\ \texttt{8},\ \texttt{2}))$

$\quad\quad \wedge \quad (\text{read-mem}\,(x,\ \text{mc-mem}\,(\text{stepn}\,(s,\ \texttt{9})),\ l)$

$\quad\quad\quad\quad = \quad \text{read-mem}\,(x,\ \text{mc-mem}\,(s),\ l)))$

THEOREM: gcd-s0-s0-2

$(\text{gcd-s0p}\,(s,\ a,\ b) \wedge (a \not\simeq \texttt{0}) \wedge (b \not\simeq \texttt{0}) \wedge (b \not< a))$

$\rightarrow \quad (\text{gcd-s0p}\,(\text{stepn}\,(s,\ \texttt{9}),\ a,\ b\ \textbf{mod}\ a)$

$\quad\quad \wedge \quad (\text{read-rn}\,(oplen,\ \texttt{14},\ \text{mc-rfile}\,(\text{stepn}\,(s,\ \texttt{9})))$

$\quad\quad\quad\quad = \quad \text{read-rn}\,(oplen,\ \texttt{14},\ \text{mc-rfile}\,(s)))$

$\quad\quad \wedge \quad (\text{linked-a6}\,(\text{stepn}\,(s,\ \texttt{9})) = \text{linked-a6}\,(s))$

$\quad\quad \wedge \quad (\text{linked-rts-addr}\,(\text{stepn}\,(s,\ \texttt{9})) = \text{linked-rts-addr}\,(s))$

$\quad\quad \wedge \quad (\text{movem-saved}\,(\text{stepn}\,(s,\ \texttt{9}),\ \texttt{4},\ \texttt{8},\ \texttt{2}) = \text{movem-saved}\,(s,\ \texttt{4},\ \texttt{8},\ \texttt{2}))$

$\quad\quad \wedge \quad (\text{read-mem}\,(x,\ \text{mc-mem}\,(\text{stepn}\,(s,\ \texttt{9})),\ l)$

$\quad\quad\quad\quad = \quad \text{read-mem}\,(x,\ \text{mc-mem}\,(s),\ l)))$

THEOREM: gcd-s0-s0-rfile-1

$(\text{gcd-s0p}\,(s,\ a,\ b) \wedge (a \not\simeq \texttt{0}) \wedge (b \not\simeq \texttt{0}) \wedge (b < a) \wedge \text{d4-7a2-5p}\,(rn))$

$\rightarrow \quad (\text{read-rn}\,(oplen,\ rn,\ \text{mc-rfile}\,(\text{stepn}\,(s,\ \texttt{9})))$

$\quad\quad = \quad \text{read-rn}\,(oplen,\ rn,\ \text{mc-rfile}\,(s)))$

THEOREM: gcd-s0-s0-rfile-2

$(\text{gcd-s0p}\,(s,\ a,\ b) \wedge (a \not\simeq \texttt{0}) \wedge (b \not\simeq \texttt{0}) \wedge (b \not< a) \wedge \text{d4-7a2-5p}\,(rn))$

$\rightarrow \quad (\text{read-rn}\,(oplen,\ rn,\ \text{mc-rfile}\,(\text{stepn}\,(s,\ \texttt{9})))$

$\quad\quad = \quad \text{read-rn}\,(oplen,\ rn,\ \text{mc-rfile}\,(s)))$

```
; put together.
```

THEOREM: gcd-s0-sn

let *sn* **be** stepn (s, gcd-t1 (a, b))
in
gcd-s0p (s, a, b)
$\rightarrow$ ((mc-status (sn) = `'running`)
  $\wedge$ (mc-pc (sn) = linked-rts-addr (s))
  $\wedge$ (iread-dn (`32`, `0`, sn) = gcd (a, b))
  $\wedge$ (read-rn (`32`, `14`, mc-rfile (sn)) = linked-a6 (s))
  $\wedge$ (read-rn (`32`, `15`, mc-rfile (sn))
    = add (`32`, read-an (`32`, `6`, s), `8`))
  $\wedge$ (read-mem (x, mc-mem (sn), k) = read-mem (x, mc-mem (s), k))) **endlet**

THEOREM: gcd-s0-sn-rfile

(gcd-s0p (s, a, b) $\wedge$ d2-7a2-5p (rn) $\wedge$ ($oplen \leq$ `32`))
$\rightarrow$ (read-rn ($oplen$, rn, mc-rfile (stepn (s, gcd-t1 (a, b))))
  = **if** d4-7a2-5p (rn) **then** read-rn ($oplen$, rn, mc-rfile (s))
    else get-vlst ($oplen$, `0`, rn, `'(2 3)`, movem-saved (s, `4`, `8`, `2`)) **endif**)

```
; the correctness of gcd.
```

THEOREM: gcd-correctness

let *sn* **be** stepn (s, gcd-t (a, b))
in
gcd-statep (s, a, b)
$\rightarrow$ ((mc-status (sn) = `'running`)
  $\wedge$ (mc-pc (sn) = rts-addr (s))
  $\wedge$ (read-rn (`32`, `14`, mc-rfile (sn))
    = read-rn (`32`, `14`, mc-rfile (s)))
  $\wedge$ (read-rn (`32`, `15`, mc-rfile (sn))
    = add (`32`, read-an (`32`, `7`, s), `4`))
  $\wedge$ ((d2-7a2-5p (rn) $\wedge$ ($oplen \leq$ `32`))
    $\rightarrow$ (read-rn ($oplen$, rn, mc-rfile (sn))
      = read-rn ($oplen$, rn, mc-rfile (s))))
  $\wedge$ (disjoint (x, k, sub (`32`, `12`, read-sp (s)), `24`)
    $\rightarrow$ (read-mem (x, mc-mem (sn), k)
      = read-mem (x, mc-mem (s), k)))
  $\wedge$ (iread-dn (`32`, `0`, sn) = gcd (a, b))) **endlet**

EVENT: Disable gcd-t.

```
; to prove that (gcd a b) does compute the greatest common divisor, we need
; to prove the following two theorems:
;     1. (gcd a b) divides a and (gcd a b) divides b.
;        i.e. (gcd a b) is a common divisor of a and b.
;     2. any divisor of a and b is no greater than (gcd a b).
```

THEOREM: remainder-remainder
$((b \textbf{ mod } c) = 0) \rightarrow (((a \textbf{ mod } b) \textbf{ mod } c) = (a \textbf{ mod } c))$

THEOREM: gcd-is-cd
$((a \textbf{ mod } \mathrm{gcd}(a,\ b)) = 0) \land ((b \textbf{ mod } \mathrm{gcd}(a,\ b)) = 0)$

THEOREM: gcd-the-greatest
$((a \not\simeq 0) \land (b \not\simeq 0) \land ((a \textbf{ mod } x) = 0) \land ((b \textbf{ mod } x) = 0))$
$\rightarrow \quad (\mathrm{gcd}(a,\ b) \not< x)$

```
; a simple timing analysis.
```

THEOREM: lessp-times-lessp
$((a < b) \land (c \not\simeq 0)) \rightarrow (a < (b * c))$

THEOREM: remainder-shrink-half
$((b \leq a) \land (b \not\simeq 0)) \rightarrow ((a \div 2) \not< (a \textbf{ mod } b))$

THEOREM: gcd-t-shrink-1
$((b \leq a) \land (b \not\simeq 0)) \rightarrow ((\log(2,\ a) - 1) \not< \log(2,\ a \textbf{ mod } b))$

THEOREM: gcd-t-shrink-2
$((b \leq a) \land (b \not\simeq 0) \land (a \neq 1))$
$\rightarrow \quad ((9 * (x + \log(2,\ a)))$
$\quad\quad \not< \quad (9 + (9 * (x + \log(2,\ a \textbf{ mod } b)))))$

DEFINITION:
gcd-t2 $(a,\ b)$
$=$ **if** $a \simeq 0$ **then** 6
elseif $b \simeq 0$ **then** 9
elseif $b < a$ **then** $9 + \text{gcd-t2}(a \textbf{ mod } b,\ b)$
elseif $a < b$ **then** $9 + \text{gcd-t2}(a,\ b \textbf{ mod } a)$
else 18 **endif**

THEOREM: gcd-t2-ub
$\text{gcd-t2}(a,\ b) \leq (18 + (9 * (\log(2,\ a) + \log(2,\ b))))$

THEOREM: gcd-t1-t2
$\text{gcd-t1}(a,\ b) = \text{gcd-t2}(a,\ b)$

THEOREM: gcd-t-ub
$\text{gcd-t}(a,\ b) \leq (22 + (9 * (\log(2,\ a) + \log(2,\ b))))$

THEOREM: gcd-t-ubound
$((a < \exp(2,\ 31)) \land (b < \exp(2,\ 31))) \rightarrow (\text{gcd-t}(a,\ b) \leq 580)$

EVENT: Make the library "gcd" and compile it.

Index