

#|

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.

|#

```
; -----
; Date:      Jan, 1991
; Modified:   Sep 4, 1992.
; File:      mc20-2.events
; -----
;
; -----
;
;                      PROVING PHASE
;
; we establish basic theory for program proving based upon our MC68020
; specification.
; -----
;
; load and compile the spec.
```

EVENT: Start with the library "mc20-1" using the compiled version.

EVENT: For efficiency, compile those definitions not yet compiled.

; a segment of the memory is ROM, iff it is pc-readable(read-only).

DEFINITION: $\text{rom-addrp}(addr, mem, n) = \text{pc-read-memp}(addr, mem, n)$

; a segment of the memory is RAM, iff it is writable(random).

DEFINITION: $\text{ram-addrp}(addr, mem, n) = \text{write-memp}(addr, mem, n)$

; x is an element of lst wrt numberp.

DEFINITION:

$\text{n-member}(x, lst)$

= **if** $lst \simeq \text{nil}$ **then** **f**
 else $(\text{fix}(x) = \text{fix}(\text{car}(lst))) \vee \text{n-member}(x, \text{cdr}(lst))$ **endif**

; mod-eq returns t iff x and y are the "same" bit-vector. It is often
; used as an induction hint to many lemmas.

DEFINITION:

$\text{mod-eq}(n, x, y)$

= **if** $n \simeq 0$ **then** **t**
 else $(\text{bcar}(x) = \text{bcar}(y)) \wedge \text{mod-eq}(n - 1, \text{bcdr}(x), \text{bcdr}(y))$ **endif**

; mod32-eq is equivalent to mod-eq with $n = 32$. But we define it as follows.

DEFINITION: $\text{mod32-eq}(x, y) = (\text{head}(x, 32) = \text{head}(y, 32))$

; the equivalence is given by this lemma.

; (prove-lemma mod-eq-lemma (rewrite)

; (equal (mod-eq 32 x y)
; (mod32-eq x y)))

; the negation of a bit-vector.

DEFINITION: $\text{neg}(n, x) = \text{sub}(n, x, 0)$

; {x, x+1, ..., x+(m-1)} and {y} are disjoint. We assume all the numbers are
; modulo 2^{32} .

DEFINITION:

$\text{disjoint0}(x, m, y)$

= **if** $m \simeq 0$ **then** **t**
 else $(\neg \text{mod32-eq}(\text{add}(32, x, m - 1), y))$
 $\wedge \text{disjoint0}(x, m - 1, y)$ **endif**

; {x, x+1, ..., x+(m-1)} and {y, y+1, ..., y+(n-1)} are disjoint.

DEFINITION:

```
disjoint( $x, m, y, n$ )  
= if  $n \simeq 0$  then t  
  else disjoint0( $x, m, \text{add}(32, y, n - 1)$ )  $\wedge$  disjoint( $x, m, y, n - 1$ ) endif  
  
;  
; BIT VECTOR AS LIST OF BIT  
; A lower level layer. This is part of my old spec that specifies the  
; microprocessor at a relatively lower level.
```

DEFINITION:

```
bit( $bv$ )  
= if  $bv \simeq \text{nil}$  then B0  
  elseif b0p(car( $bv$ )) then B0  
  else B1 endif
```

DEFINITION:

```
vec( $bv$ )  
= if  $bv \simeq \text{nil}$  then nil  
  else cdr( $bv$ ) endif
```

DEFINITION:

```
fix-bv( $bv$ )  
= if  $bv \simeq \text{nil}$  then nil  
  else cons(bit( $bv$ ), fix-bv(vec( $bv$ ))) endif
```

DEFINITION:

```
bv-len( $bv$ )  
= if  $bv \simeq \text{nil}$  then 0  
  else 1 + bv-len(vec( $bv$ )) endif
```

DEFINITION:

```
bv-not( $bv$ )  
= if  $bv \simeq \text{nil}$  then nil  
  else cons(b-not(bit( $bv$ )), bv-not(vec( $bv$ ))) endif
```

DEFINITION:

```
bv-head( $bv, n$ )  
= if  $n \simeq 0$  then nil  
  else cons(bit( $bv$ ), bv-head(vec( $bv$ ),  $n - 1$ )) endif
```

DEFINITION:

```
bv-tail( $bv, n$ )  
= if  $n \simeq 0$  then fix-bv( $bv$ )  
  else bv-tail(vec( $bv$ ),  $n - 1$ ) endif
```

DEFINITION:

```
bv-bitn (bv, n)
=  if n  $\simeq$  0 then bit (bv)
   else bv-bitn (vec (bv), n - 1) endif
```

DEFINITION:

```
bv-mbit (bv)
=  if bv  $\simeq$  nil then B0
   elseif vec (bv)  $\simeq$  nil then bit (bv)
   else bv-mbit (vec (bv)) endif
```

DEFINITION:

```
bv-adder (c, x, y)
=  if x  $\simeq$  nil then nil
   else cons (b-eor (b-eor (bit (x), bit (y)), c),
              bv-adder (b-or (b-and (bit (x), bit (y)),
                                b-or (b-and (bit (x), c), b-and (bit (y), c))),
                      vec (x),
                      vec (y))) endif
```

; conversion functions.

DEFINITION:

```
bv-to-nat (bv)
=  if bv  $\simeq$  nil then 0
   else bit (bv) + (2 * bv-to-nat (vec (bv))) endif
```

DEFINITION:

```
bv-sized-to-nat (bv, size)
=  if size  $\simeq$  0 then 0
   else bit (bv) + (2 * bv-sized-to-nat (vec (bv), size - 1)) endif
```

DEFINITION:

```
nat-to-bv (n)
=  if n  $\simeq$  0 then nil
   else cons (n mod 2, nat-to-bv (n  $\div$  2)) endif
```

DEFINITION:

```
nat-to-bv-sized (n, size)
=  if size  $\simeq$  0 then nil
   else cons (n mod 2, nat-to-bv-sized (n  $\div$  2, size - 1)) endif
```

; LISTP

THEOREM: bv-len-listp

(bv-len (x) = 0) = (x $\simeq$ nil)

THEOREM: bv-head-listp
 $\text{listp}(\text{bv-head}(x, n)) = (n \neq 0)$

THEOREM: bv-adder-listp
 $\text{listp}(\text{bv-adder}(c, x, y)) = \text{listp}(x)$

; BV-LEN

THEOREM: bv-head-len
 $\text{bv-len}(\text{bv-head}(x, n)) = \text{fix}(n)$

THEOREM: nat-to-bv-sized-len
 $\text{bv-len}(\text{nat-to-bv-sized}(x, n)) = \text{fix}(n)$

THEOREM: bv-adder-len
 $\text{bv-len}(\text{bv-adder}(c, x, y)) = \text{bv-len}(x)$

; CONVERSION THEOREMS

; nat-to-bv-sized only considers the first k binary digits of natural number n.

THEOREM: nat-to-bv-sized-la0
 $\text{nat-to-bv-sized}(n \bmod \exp(2, k), k) = \text{nat-to-bv-sized}(n, k)$

; rewrite nat-to-bv-sized into nat-to-bv.

THEOREM: nat-to-bv-sized-head
 $\text{nat-to-bv-sized}(m, n) = \text{bv-head}(\text{nat-to-bv}(m), n)$

; rewrite bv-sized-to-nat into bv-to-nat.

THEOREM: bv-sized-to-nat-head
 $\text{bv-sized-to-nat}(x, n) = \text{bv-to-nat}(\text{bv-head}(x, n))$

THEOREM: bv-to-nat-to-bv
 $\text{bv-head}(\text{nat-to-bv}(\text{bv-to-nat}(x)), n) = \text{bv-head}(x, n)$

THEOREM: nat-to-bv-to-nat
 $\text{bv-to-nat}(\text{nat-to-bv}(n)) = \text{fix}(n)$

THEOREM: bv-head-nat
 $\text{bv-to-nat}(\text{bv-head}(x, n))$
 $= \text{if } \text{bv-to-nat}(x) < \exp(2, n) \text{ then } \text{bv-to-nat}(x)$
 $\text{else } \text{bv-to-nat}(x) \bmod \exp(2, n) \text{ endif}$

THEOREM: bv-to-nat-to-bv-sized
 $\text{nat-to-bv-sized}(\text{bv-to-nat}(x), n) = \text{bv-head}(x, n)$

THEOREM: nat-to-bv-sized-to-nat
 bv-to-nat (nat-to-bv-sized (m , n))
 = **if** $m < \text{exp}(2, n)$ **then** fix (m)
 else $m \bmod \text{exp}(2, n)$ **endif**

THEOREM: bv-sized-to-nat-to-bv-sized
 nat-to-bv-sized (bv-sized-to-nat (x , n), n) = bv-head (x , n)

THEOREM: nat-to-bv-sized-sized-to-nat
 bv-sized-to-nat (nat-to-bv-sized (m , n), n)
 = **if** $m < \text{exp}(2, n)$ **then** fix (m)
 else $m \bmod \text{exp}(2, n)$ **endif**

EVENT: Disable bv-sized-to-nat-head.

EVENT: Disable nat-to-bv-sized-head.

; BV-BITN and BV-NOT

THEOREM: bv-bitn-not
 bv-bitn (bv-not (x), i)
 = **if** $i < \text{bv-len}(x)$ **then** b-not (bv-bitn (x , i))
 else B0 **endif**

THEOREM: bv-to-nat-range
 nat-range (bv-to-nat (x), bv-len (x))

; BV-ADDER
 ; a bridge function.

DEFINITION:
 bv-adder&carry (c , x , y)
 = **if** $x \simeq \text{nil}$ **then** cons (fix-bit (c), **nil**)
 else cons (b-eor (c , b-eor (bit (x), bit (y))),
 bv-adder&carry (b-or (b-and (bit (x), bit (y)),
 b-or (b-and (bit (x), c),
 b-and (bit (y), c))),
 vec (x),
 vec (y))) **endif**

THEOREM: bv-adder&carry-len
 bv-len (bv-adder&carry (c , x , y)) = (1 + bv-len (x))

THEOREM: fix-bv-adder&carry
 fix-bv (bv-adder&carry (c , x , y)) = bv-adder&carry (c , x , y)

; the relation between bv-adder&carry and bv-adder. We have successfully
 ; constructed the bridge!

THEOREM: bv-adder-bridge

$$\text{bv-adder}(c, x, y) = \text{bv-head}(\text{bv-adder\&carry}(c, x, y), \text{bv-len}(x))$$

; lifting lemmas

THEOREM: bv-not-lognot

$$\text{bv-to-nat}(\text{bv-not}(x)) = \text{lognot}(\text{bv-len}(x), \text{bv-to-nat}(x))$$

THEOREM: bv-bitn-bitn

$$\text{bv-bitn}(x, n) = \text{bitn}(\text{bv-to-nat}(x), n)$$

THEOREM: bv-mbit-bitn

$$\begin{aligned} &\text{bv-mbit}(x) \\ &= \text{if } \text{bv-len}(x) = 0 \text{ then } 0 \\ &\quad \text{else } \text{bitn}(\text{bv-to-nat}(x), \text{bv-len}(x) - 1) \text{ endif} \end{aligned}$$

THEOREM: bv-adder&carry-nat

$$\begin{aligned} &(\text{bv-len}(x) = \text{bv-len}(y)) \\ &\rightarrow (\text{bv-to-nat}(\text{bv-adder\&carry}(c, x, y)) \\ &\quad = (\text{fix-bit}(c) + \text{bv-to-nat}(x) + \text{bv-to-nat}(y))) \end{aligned}$$

THEOREM: bv-adder-nat

$$\begin{aligned} &((\text{bv-len}(x) = \text{bv-len}(y)) \wedge \text{bitp}(c)) \\ &\rightarrow (\text{bv-to-nat}(\text{bv-adder}(c, x, y)) \\ &\quad = \text{adder}(\text{bv-len}(x), c, \text{bv-to-nat}(x), \text{bv-to-nat}(y))) \end{aligned}$$

; EVENP

; add-evenp states that the sum of two even numbers is still even.

THEOREM: add-evenp

$$(\text{evenp}(x) \wedge \text{evenp}(y)) \rightarrow \text{evenp}(\text{add}(n, x, y))$$

EVENT: Disable evenp.

; LOGNOT

THEOREM: lognot-0

$$\begin{aligned} &(x \notin \mathbf{N}) \\ &\rightarrow ((\text{lognot}(x, y) = \text{lognot}(0, y)) \wedge (\text{lognot}(y, x) = \text{lognot}(y, 0))) \end{aligned}$$

THEOREM: lognot-nat-range

$$\text{nat-range}(\text{lognot}(n, x), n)$$

THEOREM: lognot-lognot
 $\text{lognot}(n, \text{lognot}(n, x)) = \text{head}(x, n)$

THEOREM: lognot-cancel
 $(\text{nat-range}(x, n) \wedge \text{nat-range}(y, n))$
 $\rightarrow ((\text{lognot}(n, x) = \text{lognot}(n, y)) = (\text{fix}(x) = \text{fix}(y)))$

; the integer interpretation of lognot.

THEOREM: lognot-int
 $(\text{nat-range}(x, n) \wedge (n \neq 0))$
 $\rightarrow (\text{nat-to-int}(\text{lognot}(n, x), n) = \text{iplus}(-1, \text{ineg}(\text{nat-to-int}(x, n))))$

THEOREM: adder-lognot
 $\text{bitp}(c)$
 $\rightarrow (\text{lognot}(n, \text{adder}(n, c, x, y))$
 $\quad = \text{adder}(n, \text{b-not}(c), \text{lognot}(n, x), \text{lognot}(n, y)))$

EVENT: Disable lognot.

; BASICS ABOUT HEAD & TAIL

THEOREM: head-0
 $\text{head}(0, n) = 0$

THEOREM: tail-0
 $\text{tail}(0, n) = 0$

THEOREM: head-of-0
 $\text{head}(x, 0) = 0$

THEOREM: tail-of-0
 $\text{tail}(x, 0) = \text{fix}(x)$

THEOREM: head-lessp
 $\text{head}(x, n) < \text{exp}(2, n)$

THEOREM: tail-lessp
 $(\text{tail}(x, n) < y) = (x < (\text{exp}(2, n) * y))$

THEOREM: head-leq
 $x \not\leq \text{head}(x, n)$

THEOREM: tail-leq
 $x \not\leq \text{tail}(x, n)$

THEOREM: tail-equal-0

$$(\text{tail}(x, n) = 0) = \text{nat-range}(x, n)$$

THEOREM: head-lemma

$$\text{nat-range}(x, n) \rightarrow (\text{head}(x, n) = \text{fix}(x))$$

THEOREM: tail-lemma

$$\text{nat-range}(x, n) \rightarrow (\text{tail}(x, n) = 0)$$

THEOREM: replace-0

$$\begin{aligned} &(\text{replace}(0, x, y) = \text{fix}(y)) \\ \wedge &(\text{replace}(n, x, 0) = \text{head}(x, n)) \\ \wedge &((\text{nat-range}(x, n) \wedge \text{nat-range}(y, n)) \\ &\rightarrow (\text{replace}(n, x, y) = \text{fix}(x))) \end{aligned}$$

THEOREM: app-0

$$\begin{aligned} &(\text{app}(n, x, 0) = \text{head}(x, n)) \\ \wedge &(\text{app}(n, 0, y) = (y * \exp(2, n))) \\ \wedge &(\text{app}(0, x, y) = \text{fix}(y)) \end{aligned}$$

; THEOREMS ABOUT SHIFT AND ROTATE
; the key events are LSL-UINT, LSR-UINT, ASL-INT, and ASR-INT.
; ASL-INT and ASR-INT are pretty hard. It took me a few days to get the
; proofs. It would be a good challenge to other provers.

THEOREM: lsl-uint

$$\begin{aligned} &\text{uint-range}(x, n - \text{cnt}) \\ \rightarrow &(\text{nat-to-uint}(\text{lsl}(n, x, \text{cnt})) = (\text{nat-to-uint}(x) * \exp(2, \text{cnt}))) \end{aligned}$$

THEOREM: lsr-uint

$$\text{nat-to-uint}(\text{lsr}(x, \text{cnt})) = (\text{nat-to-uint}(x) \div \exp(2, \text{cnt}))$$

; two lemmas for asl-int.

THEOREM: remainder-diff-la

$$\begin{aligned} &(((x \bmod z) = 0) \wedge (y < z)) \\ \rightarrow &(((x - y) \bmod z) \\ &= \text{if } x \leq y \text{ then } 0 \\ &\text{elseif } y \simeq 0 \text{ then } 0 \\ &\text{else } z - y \text{ endif}) \end{aligned}$$

THEOREM: asl-int-crock1

$$\begin{aligned} &(x < \exp(2, (n - s) - 1)) \\ \rightarrow &(\text{head}(x * \exp(2, s), n) = (x * \exp(2, s))) \end{aligned}$$

THEOREM: asl-int-crock2

$$\begin{aligned} & (((\exp(2, s) * (\exp(2, n) - x)) \leq \exp(2, n - 1)) \wedge (x < \exp(2, n))) \\ \rightarrow & \text{head}(x * \exp(2, s), n) \\ & = (\exp(2, n) - (\exp(2, s) * (\exp(2, n) - x))) \end{aligned}$$

THEOREM: asl-int

$$\begin{aligned} & (\text{nat-rangep}(x, n) \wedge \text{int-rangep}(\text{nat-to-int}(x, n), n - s)) \\ \rightarrow & (\text{nat-to-int}(\text{asl}(n, x, s), n) = \text{itimes}(\text{nat-to-int}(x, n), \exp(2, s))) \end{aligned}$$

EVENT: Disable asl-int-crock1.

EVENT: Disable asl-int-crock2.

THEOREM: quotient-diff-la

$$\begin{aligned} & (((x \bmod z) = 0) \wedge (y < z)) \\ \rightarrow & (((x - y) \div z) \\ & = \text{if } y \simeq 0 \text{ then } x \div z \\ & \text{else } (x \div z) - 1 \text{ endif}) \end{aligned}$$

THEOREM: remainder-diff

$$\begin{aligned} & ((x \bmod z) = 0) \\ \rightarrow & (((x - y) \bmod z) \\ & = \text{if } (x \leq y) \vee ((y \bmod z) = 0) \text{ then } 0 \\ & \text{else } z - (y \bmod z) \text{ endif}) \end{aligned}$$

THEOREM: quotient-diff

$$\begin{aligned} & ((x \bmod z) = 0) \\ \rightarrow & (((x - y) \div z) \\ & = \text{if } (y \bmod z) = 0 \text{ then } (x \div z) - (y \div z) \\ & \text{else } ((x \div z) - (y \div z)) - 1 \text{ endif}) \end{aligned}$$

THEOREM: quotient-exp-lessp

$$\begin{aligned} & (cnt \leq n) \\ \rightarrow & (((x \div \exp(2, cnt)) < \exp(2, n - cnt)) = (x < \exp(2, n))) \end{aligned}$$

THEOREM: lessp-times-exp-1s

$$\begin{aligned} & ((m + n) = \text{fix}(k)) \\ \rightarrow & (((x + (\exp(2, m) * (\exp(2, n) - 1))) < \exp(2, k - 1)) \\ & = \text{if } n \simeq 0 \text{ then } x < \exp(2, (m + n) - 1) \\ & \text{else f endif}) \end{aligned}$$

THEOREM: lessp-app-1s

$$\begin{aligned} & ((m + n) = \text{fix}(k)) \\ \rightarrow & ((\text{app}(m, x, \exp(2, n) - 1) < \exp(2, k - 1)) \\ & = \text{if } n \simeq 0 \text{ then head}(x, m) < \exp(2, (m + n) - 1) \\ & \text{else f endif}) \end{aligned}$$

THEOREM: times-sub1

$$((y - 1) * x) = ((x * y) - x)$$

THEOREM: difference-app-1s

$$((m + n) = \text{fix}(k))$$

$$\rightarrow ((\text{exp}(2, k) - \text{app}(m, x, \text{exp}(2, n) - 1)) = (\text{exp}(2, m) - \text{head}(x, m)))$$

EVENT: Disable times-sub1.

THEOREM: asr-int-crock

$$((x < \text{exp}(2, n)) \wedge (x \not< \text{exp}(2, n - 1)) \wedge (cnt \leq n))$$

$$\rightarrow ((\text{app}(n - cnt, x \div \text{exp}(2, cnt), \text{exp}(2, cnt) - 1) < \text{exp}(2, n - 1)) \\ = \text{f})$$

THEOREM: asr-int

$$\text{nat-range}(x, n)$$

$$\rightarrow (\text{nat-to-int}(\text{asr}(n, x, i), n)$$

$$= \text{if negative}(\text{nat-to-int}(x, n))$$

$$\text{then if remainder}(\text{nat-to-int}(x, n), \text{exp}(2, i)) = 0$$

$$\text{then quotient}(\text{nat-to-int}(x, n), \text{exp}(2, i))$$

$$\text{else iplus}(-1, \text{quotient}(\text{nat-to-int}(x, n), \text{exp}(2, i))) \text{ endif}$$

$$\text{else quotient}(\text{nat-to-int}(x, n), \text{exp}(2, i)) \text{ endif})$$

EVENT: Disable asr-int-crock.

EVENT: Disable quotient-exp-lessp.

EVENT: Disable remainder-diff.

EVENT: Disable quotient-diff.

THEOREM: lsl-0

$$\text{lsl}(n, x, 0) = \text{head}(x, n)$$

THEOREM: lsl-nat-range

$$\text{nat-range}(\text{lsl}(n, x, cnt), n)$$

THEOREM: lsl-lsl

$$\text{lsl}(n, \text{lsl}(n, x, cnt1), cnt2) = \text{lsl}(n, x, cnt1 + cnt2)$$

THEOREM: lsr-0

$$\text{lsr}(x, 0) = \text{fix}(x)$$

THEOREM: lsr-nat-range
 $\text{nat-range}(x, n) \rightarrow \text{nat-range}(\text{lsr}(x, \text{cnt}), n)$

THEOREM: lsr-lsr
 $\text{lsr}(\text{lsr}(x, \text{cnt1}), \text{cnt2}) = \text{lsr}(x, \text{cnt1} + \text{cnt2})$

THEOREM: asl-0
 $\text{asl}(n, x, 0) = \text{head}(x, n)$

THEOREM: asl-nat-range
 $\text{nat-range}(\text{asl}(n, x, \text{cnt}), n)$

THEOREM: asl-asl
 $\text{asl}(n, \text{asl}(n, x, \text{cnt1}), \text{cnt2}) = \text{asl}(n, x, \text{cnt1} + \text{cnt2})$

THEOREM: asr-0
 $\text{nat-range}(x, n) \rightarrow (\text{asr}(n, x, 0) = \text{fix}(x))$

THEOREM: asr-nat-range
 $\text{nat-range}(x, n) \rightarrow \text{nat-range}(\text{asr}(n, x, \text{cnt}), n)$

```
; unproved! Not useful yet.
; (prove-lemma asr-asr (rewrite)
;   (implies (and (nat-range x n)
;                 (not (zerop n)))
;             (equal (asr n (asr n x cnt1) cnt2)
;                   (asr n x (plus cnt1 cnt2)))))
```

; the integer interpretation of ext.

THEOREM: mbit-means-lessp
 $\text{nat-range}(x, n)$
 $\rightarrow (\text{bitn}(x, n - 1)$
 $\quad = \text{if } x < \text{exp}(2, n - 1) \text{ then } 0$
 $\quad \text{else } 1 \text{ endif})$

THEOREM: exp2-lessp
 $(i < j) \rightarrow (\text{exp}(2, i) < \text{exp}(2, j))$

; note that ext-lemma takes care of the other aspects about ext.

THEOREM: ext-int
 $(\text{nat-range}(x, n) \wedge (n < \text{size}))$
 $\rightarrow (\text{nat-to-int}(\text{ext}(n, x, \text{size}), \text{size}) = \text{nat-to-int}(x, n))$

EVENT: Disable exp2-lessp.

```

;                                INTEGERS!
; functions with a postfix 'end' will eventually be disabled. We use them
; to 'tell' the prover that something unexpected happens. They can
; somehow force the prover to stop proving and print out some information.

```

DEFINITION:

```

adder-int-end (n, c, x, y)
=  nat-to-int ((fix-bit (c) + int-to-nat (x, n) + int-to-nat (y, n))
               mod exp (2, n),
               n)

```

THEOREM: plus-to-iplus

```

(integerp (x) ∧ integerp (y) ∧ int-range (x, n) ∧ int-range (y, n))
→  (adder-int-end (n, c, x, y)
    =  if int-range (iplus (x, iplus (y, fix-bit (c))), n)
        then iplus (x, iplus (y, fix-bit (c)))
        elseif negativep (x)
        then iplus (x, iplus (y, iplus (fix-bit (c), exp (2, n))))
        else iplus (x, iplus (y, iplus (fix-bit (c), - exp (2, n)))) endif)

```

EVENT: Disable plus-to-iplus.

THEOREM: iplus-with-carry-negativep

```

(integerp (x) ∧ integerp (y) ∧ bitp (z) ∧ negativep (x) ∧ negativep (y))
→  negativep (iplus (x, iplus (y, z)))

```

THEOREM: iplus-with-carry-non-negativep

```

(integerp (x)
 ∧ integerp (y)
 ∧ bitp (z)
 ∧ (¬ negativep (x))
 ∧ (¬ negativep (y)))
→  (¬ negativep (iplus (x, iplus (y, z))))

```

; two lemmas for addx-v. They will be disabled right after we use them.

THEOREM: addx-v-crock1

```

(integerp (x)
 ∧ integerp (y)
 ∧ bitp (z)
 ∧ int-range (x, n)
 ∧ int-range (y, n))
→  (¬ negativep (iplus (x, iplus (y, iplus (z, exp (2, n))))))

```

THEOREM: addx-v-crock2

$$\begin{aligned}
& (\text{integerp}(x) \\
& \wedge \text{integerp}(y) \\
& \wedge \text{bitp}(z) \\
& \wedge (n \neq 0) \\
& \wedge \text{int-rangep}(x, n) \\
& \wedge \text{int-rangep}(y, n)) \\
& \rightarrow \text{negativep}(\text{iplus}(x, \text{iplus}(y, \text{iplus}(z, -\text{exp}(2, n)))))
\end{aligned}$$

; two lemmas for BMI of add. They will be disabled right after we
; use them.

THEOREM: add-bmi-crock1

$$\begin{aligned}
& (\text{integerp}(x) \wedge \text{integerp}(y) \wedge \text{int-rangep}(x, n) \wedge \text{int-rangep}(y, n)) \\
& \rightarrow (\neg \text{negativep}(\text{iplus}(x, \text{iplus}(y, \text{exp}(2, n)))))
\end{aligned}$$

THEOREM: add-bmi-crock2

$$\begin{aligned}
& (\text{integerp}(x) \wedge \text{integerp}(y) \wedge \text{int-rangep}(x, n) \wedge \text{int-rangep}(y, n)) \\
& \rightarrow \text{negativep}(\text{iplus}(x, \text{iplus}(y, -\text{exp}(2, n)))))
\end{aligned}$$

; two lemmas for BMI of sub. They will be disabled right after we
; use them.

THEOREM: illessp-crock1

$$\begin{aligned}
& (\text{integerp}(x) \wedge \text{integerp}(y) \wedge \text{int-rangep}(x, n) \wedge \text{int-rangep}(y, n)) \\
& \rightarrow (\neg \text{illessp}(\text{iplus}(y, \text{exp}(2, n)), x))
\end{aligned}$$

THEOREM: illessp-crock2

$$\begin{aligned}
& (\text{integerp}(x) \wedge \text{integerp}(y) \wedge \text{int-rangep}(x, n) \wedge \text{int-rangep}(y, n)) \\
& \rightarrow \text{illessp}(\text{iplus}(y, -\text{exp}(2, n)), x)
\end{aligned}$$

; the addition of two opposite sign integers will never overflow, provided
; the two integers are in "good" range.

THEOREM: iplus-int-range1

$$\begin{aligned}
& (\text{integerp}(x) \\
& \wedge \text{integerp}(y) \\
& \wedge \text{bitp}(z) \\
& \wedge \text{int-rangep}(x, n) \\
& \wedge \text{int-rangep}(y, n) \\
& \wedge (\neg \text{negativep}(x)) \\
& \wedge \text{negativep}(y)) \\
& \rightarrow \text{int-rangep}(\text{iplus}(x, \text{iplus}(y, z)), n)
\end{aligned}$$

THEOREM: iplus-int-range2

$$\begin{aligned}
& (\text{integerp } (x)) \\
& \wedge \text{ integerp } (y) \\
& \wedge \text{ bitp } (z) \\
& \wedge \text{ int-rangep } (x, n) \\
& \wedge \text{ int-rangep } (y, n) \\
& \wedge \text{ negativep } (x) \\
& \wedge (\neg \text{ negativep } (y)) \\
& \rightarrow \text{ int-rangep } (\text{iplus } (x, \text{iplus } (y, z)), n)
\end{aligned}$$

THEOREM: idifference-int-range1

$$\begin{aligned}
& (\text{int-rangep } (x, n)) \\
& \wedge \text{ int-rangep } (y, n) \\
& \wedge (\neg \text{ ilessp } (x, y)) \\
& \wedge (\neg \text{ negativep } (y)) \\
& \rightarrow \text{ int-rangep } (\text{idifference } (y, x), n)
\end{aligned}$$

THEOREM: idifference-int-range2

$$\begin{aligned}
& (\text{integerp } (x)) \\
& \wedge \text{ integerp } (y) \\
& \wedge \text{ int-rangep } (x, n) \\
& \wedge \text{ int-rangep } (y, n) \\
& \wedge (\neg \text{ ilessp } (y, x)) \\
& \wedge \text{ negativep } (y) \\
& \rightarrow \text{ int-rangep } (\text{idifference } (y, x), n)
\end{aligned}$$

; two lemmas about uint-rangep and int-rangep. We introduce them before any
; lemmas about uint-range and int-range.

THEOREM: uint-rangep-la

$$(x < \exp(2, n)) \rightarrow \text{uint-rangep } (x, n)$$

THEOREM: int-rangep-la

$$((x \in \mathbf{N}) \wedge (x < \exp(2, n - 1))) \rightarrow \text{int-rangep } (x, n)$$

; NAT-TO-UINT & UINT-TO-NAT

THEOREM: nat-to-uint-to-nat

$$\text{uint-to-nat } (\text{nat-to-uint } (x)) = \text{fix } (x)$$

THEOREM: uint-to-nat-to-uint

$$\text{nat-to-uint } (\text{uint-to-nat } (x)) = \text{fix } (x)$$

THEOREM: uint-to-nat-rangep

$$(x < \exp(2, n)) \rightarrow \text{nat-rangep } (\text{uint-to-nat } (x), n)$$

THEOREM: nat-to-uint-range
 $\text{nat-range}(x, n) \rightarrow \text{uint-range}(\text{nat-to-uint}(x), n)$

; NAT-TO-INT & INT-TO-NAT

THEOREM: nat-to-int-0
 $\text{nat-to-int}(0, n) = 0$

THEOREM: int-to-nat-0
 $\text{int-to-nat}(0, n) = 0$

; nat-to-int always returns an integer, provided the input is a "good"
 ; natural number.

THEOREM: nat-to-int-integer
 $\text{integer}(\text{nat-to-int}(x, n)) = \text{nat-range}(x, n)$

; nat-to-int always returns a "good" integer.

THEOREM: nat-to-int-range
 $\text{int-range}(\text{nat-to-int}(x, n), n)$

; int-to-nat always returns a "good" natural number, provided the input
 ; is a "good" integer.

THEOREM: int-to-nat-range
 $(\text{integer}(x) \wedge \text{int-range}(x, n)) \rightarrow \text{nat-range}(\text{int-to-nat}(x, n), n)$

THEOREM: nat-to-int-to-nat
 $\text{nat-range}(x, n) \rightarrow (\text{int-to-nat}(\text{nat-to-int}(x, n), n) = \text{fix}(x))$

THEOREM: int-to-nat-to-int
 $(\text{integer}(x) \wedge \text{int-range}(x, n))$
 $\rightarrow (\text{nat-to-int}(\text{int-to-nat}(x, n), n) = \text{fix-int}(x))$

THEOREM: int-to-nat=0
 $(\text{int-range}(x, n) \wedge \text{integer}(x))$
 $\rightarrow ((\text{int-to-nat}(x, n) = 0) = (x = 0))$

; /x/ = /y/ --> x = y.

THEOREM: nat-to-int=
 $(\text{nat-range}(x, n) \wedge \text{nat-range}(y, n))$
 $\rightarrow ((\text{nat-to-int}(x, n) = \text{nat-to-int}(y, n)) = (\text{fix}(x) = \text{fix}(y)))$

; nat-to-int and int-to-nat are disabled.

EVENT: Disable nat-to-int.

EVENT: Disable int-to-nat.

; theorems about int-rangep.

THEOREM: abs-lessp-int-rangep
 $(\text{int-rangep}(x, n) \wedge (\text{abs}(y) < \text{abs}(x))) \rightarrow \text{int-rangep}(y, n)$

; 0 is always in "good" integer range.

THEOREM: int-rangep-of-0
 $\text{int-rangep}(0, n)$

THEOREM: sub1-int-rangep
 $\text{int-rangep}(x, n) \rightarrow \text{int-rangep}(x - 1, n)$

THEOREM: difference-int-rangep
 $\text{int-rangep}(x, n) \rightarrow \text{int-rangep}(x - y, n)$

THEOREM: quotient-int-rangep
 $\text{int-rangep}(x, n) \rightarrow \text{int-rangep}(x \div y, n)$

THEOREM: remainder-int-rangep
 $(\text{int-rangep}(y, n) \wedge (y \neq 0)) \rightarrow \text{int-rangep}(x \bmod y, n)$

; iquotient is almost always in the integer range.

THEOREM: iquotient-int-rangep
 $\text{int-rangep}(x, n)$
 $\rightarrow (\text{int-rangep}(\text{iquotient}(x, y), n)$
 $\quad = \text{if } y = -1 \text{ then } x \neq (-\text{exp}(2, n - 1))$
 $\quad \text{else t endif})$

; iremainder is almost always in the integer range.

THEOREM: iremainder-int-rangep
 $(\text{int-rangep}(y, n) \wedge \text{integerp}(y) \wedge (y \neq 0))$
 $\rightarrow \text{int-rangep}(\text{iremainder}(x, y), n)$

; lemmas about integerp.

THEOREM: integerp-fix-int
 $\text{integerp}(x) \rightarrow (\text{fix-int}(x) = x)$

THEOREM: fix-int-integerp
 $\text{integerp}(\text{fix-int}(x))$

THEOREM: minus-integerp
 $\text{integerp}(-x) = (x \neq 0)$

THEOREM: integerp-minus0
 $\text{integerp}(x) \rightarrow (x \neq (-0))$

THEOREM: negativep-guts0
 $(\text{integerp}(x) \wedge \text{negativep}(x)) \rightarrow (\text{negative-guts}(x) \neq 0)$

THEOREM: numberp-integerp
 $(x \in \mathbf{N}) \rightarrow \text{integerp}(x)$

THEOREM: iplus-integerp
 $\text{integerp}(\text{iplus}(x, y))$

THEOREM: ineg-integerp
 $\text{integerp}(\text{ineg}(x))$

THEOREM: idifference-integerp
 $\text{integerp}(\text{idifference}(x, y))$

THEOREM: itimes-integerp
 $\text{integerp}(\text{itimes}(x, y))$

THEOREM: iremainder-integerp
 $\text{integerp}(\text{iremainder}(x, y))$

THEOREM: iquotient-integerp
 $\text{integerp}(\text{iquotient}(x, y))$

; theorems about iplus, idifference, and ineg.

THEOREM: iplus-commutativity
 $\text{iplus}(x, y) = \text{iplus}(y, x)$

THEOREM: iplus-commutativity1
 $\text{iplus}(x, \text{iplus}(y, z)) = \text{iplus}(y, \text{iplus}(x, z))$

THEOREM: iplus-associativity
 $\text{iplus}(\text{iplus}(x, y), z) = \text{iplus}(x, \text{iplus}(y, z))$

THEOREM: ineg-iplus
 $\text{ineg}(\text{iplus}(x, y)) = \text{iplus}(\text{ineg}(x), \text{ineg}(y))$

THEOREM: iplus-0
 $\text{izerop}(x) \rightarrow ((\text{iplus}(x, y) = \text{fix-int}(y)) \wedge (\text{iplus}(y, x) = \text{fix-int}(y)))$

THEOREM: iplus-1-1
 $\text{iplus}(1, \text{iplus}(-1, x)) = \text{fix-int}(x)$

THEOREM: idifference-x-x
 $\text{idifference}(x, x) = 0$

THEOREM: idifference-negativep
 $\text{negativep}(\text{idifference}(x, y)) = \text{ilessp}(x, y)$

; lemmas about illessp.

THEOREM: illessp-reflex
 $\neg \text{ilessp}(x, x)$

THEOREM: illessp-entails-ileq
 $\text{ilessp}(x, y) \rightarrow (\neg \text{ilessp}(y, x))$

; lemmas about itimes.

THEOREM: itimes-0
 $\text{itimes}(0, x) = 0$

THEOREM: itimes-equal-0
 $(\text{itimes}(x, y) = 0) = (\text{izerop}(x) \vee \text{izerop}(y))$

THEOREM: itimes-commutativity
 $\text{itimes}(x, y) = \text{itimes}(y, x)$

THEOREM: itimes-associativity
 $\text{itimes}(\text{itimes}(x, y), z) = \text{itimes}(x, \text{itimes}(y, z))$

THEOREM: itimes-equal-cancellation
 $(\neg \text{izerop}(x))$
 $\rightarrow ((\text{itimes}(x, y) = \text{itimes}(x, z)) = (\text{fix-int}(y) = \text{fix-int}(z)))$

THEOREM: itimes-sign
 $(\text{integerp}(x) \wedge \text{integerp}(y))$
 $\rightarrow (\text{negativep}(\text{itimes}(x, y))$
 $\quad = (((x \in \mathbf{N}) \wedge (x \neq 0) \wedge \text{negativep}(y))$
 $\quad \vee (\text{negativep}(x) \wedge (y \in \mathbf{N}) \wedge (y \neq 0)))$

; lemmas about iremainder and iquotient.

THEOREM: iremainder-wrt-1
 $\text{iremainder}(x, 1) = 0$

THEOREM: iquotient-wrt-1
 $\text{iquotient}(x, 1) = \text{fix-int}(x)$

THEOREM: iremainder-wrt-1
 $\text{iremainder}(x, -1) = 0$

THEOREM: iquotient-wrt-1
 $\text{iquotient}(x, -1) = \text{ineg}(x)$

EVENT: Disable iplus.

EVENT: Disable itimes.

EVENT: Disable iremainder.

EVENT: Disable iquotient.

EVENT: Disable integerp.

; unsigned interpretation of mulu.

THEOREM: times-lessp-1
 $((x < x1) \wedge (y < y1)) \rightarrow ((x * y) < (x1 * y1))$

; three instances of mulu-nat.

THEOREM: mulu_1632-nat
 $(\text{nat-range}(x, 16) \wedge \text{nat-range}(y, 16))$
 $\rightarrow (\text{nat-to-uint}(\text{mulu}(32, x, y, 16)))$
 $= (\text{nat-to-uint}(x) * \text{nat-to-uint}(y))$

THEOREM: mulu_3264-nat
 $(\text{nat-range}(x, 32) \wedge \text{nat-range}(y, 32))$
 $\rightarrow (\text{nat-to-uint}(\text{mulu}(64, x, y, 32)))$
 $= (\text{nat-to-uint}(x) * \text{nat-to-uint}(y))$

THEOREM: mulu_3232-nat
 $\text{uint-range}(\text{nat-to-uint}(x) * \text{nat-to-uint}(y), 32)$
 $\rightarrow (\text{nat-to-uint}(\text{mulu}(32, x, y, 32)))$
 $= (\text{nat-to-uint}(x) * \text{nat-to-uint}(y))$

; signed interpretation of muls.

THEOREM: exp2-lessp-crock

$$\begin{aligned} & ((i \neq 0) \\ & \wedge \text{ if } k < i \text{ then f} \\ & \quad \text{else t endif}) \\ & \rightarrow (\exp(2, i - 1) < \exp(2, k)) \end{aligned}$$

THEOREM: head-int-crock

$$\begin{aligned} & (\text{integerp}(x) \wedge \text{int-rangep}(x, n) \wedge (n \leq j)) \\ & \rightarrow (\text{head}(\text{int-to-nat}(x, j), n) = \text{int-to-nat}(x, n)) \end{aligned}$$

EVENT: Disable exp2-lessp-crock.

THEOREM: times-lessp_2

$$\begin{aligned} & ((x \leq x1) \wedge (y \leq y1) \wedge (x1 \neq 0) \wedge (y1 \neq 0)) \\ & \rightarrow ((x * y) < (2 * x1 * y1)) \end{aligned}$$

THEOREM: times-lessp_3

$$\begin{aligned} & ((x \leq \exp(2, n - 1)) \wedge (y \leq \exp(2, n - 1)) \wedge (n \neq 0)) \\ & \rightarrow (((x * y) < \exp(2, (n + n) - 1)) = \mathbf{t}) \end{aligned}$$

THEOREM: times-lessp_4

$$\begin{aligned} & ((x \leq \exp(2, n - 1)) \wedge (y \leq \exp(2, n - 1)) \wedge (n \neq 0)) \\ & \rightarrow ((\exp(2, (n + n) - 1) < (x * y)) = \mathbf{f}) \end{aligned}$$

THEOREM: muls-crock

$$\begin{aligned} & (\text{int-rangep}(x, n) \wedge \text{int-rangep}(y, n) \wedge (n \neq 0)) \\ & \rightarrow \text{int-rangep}(\text{itimes}(x, y), n + n) \end{aligned}$$

; three instances of muls-int.

THEOREM: muls_1632-int

$$\begin{aligned} & (\text{nat-rangep}(x, 16) \wedge \text{nat-rangep}(y, 16)) \\ & \rightarrow (\text{nat-to-int}(\text{muls}(32, x, y, 16), 32) \\ & \quad = \text{itimes}(\text{nat-to-int}(x, 16), \text{nat-to-int}(y, 16))) \end{aligned}$$

THEOREM: muls_3264-int

$$\begin{aligned} & (\text{nat-rangep}(x, 32) \wedge \text{nat-rangep}(y, 32)) \\ & \rightarrow (\text{nat-to-int}(\text{muls}(64, x, y, 32), 64) \\ & \quad = \text{itimes}(\text{nat-to-int}(x, 32), \text{nat-to-int}(y, 32))) \end{aligned}$$

THEOREM: muls_3232-int

$$\begin{aligned} & \text{int-rangep}(\text{itimes}(\text{nat-to-int}(x, 32), \text{nat-to-int}(y, 32)), 32) \\ & \rightarrow (\text{nat-to-int}(\text{muls}(32, x, y, 32), 32) \\ & \quad = \text{itimes}(\text{nat-to-int}(x, 32), \text{nat-to-int}(y, 32))) \end{aligned}$$

EVENT: Disable times-lessp_1.

EVENT: Disable times-lessp_2.

EVENT: Disable times-lessp_3.

EVENT: Disable times-lessp_4.

EVENT: Disable muls-crock.

```
; signed interpretation of irem.  
; notice that we only consider the case that the divisor and the result  
; have the same boundary n. Since it is enough to cover the situations  
; in the DIVS instructions.
```

THEOREM: irem-int
 $(\text{nat-range}(x, n) \wedge (\text{nat-to-int}(x, n) \neq 0))$
 $\rightarrow (\text{nat-to-int}(\text{irem}(n, x, n, y, j), n)$
 $\quad = \text{remainder}(\text{nat-to-int}(y, j), \text{nat-to-int}(x, n)))$

```
; signed interpretation of iquot.  
; notice that we only consider the case that the length of the result  
; is at most the length of the dividend. Since it is enough to cover  
; the situations in DIVS instruction.
```

THEOREM: iquot-int
 $(\text{int-range}(\text{iquot}(\text{nat-to-int}(y, j), \text{nat-to-int}(x, i)), n) \wedge (n \leq j))$
 $\rightarrow (\text{nat-to-int}(\text{iquot}(n, x, i, y, j), n)$
 $\quad = \text{iquot}(\text{nat-to-int}(y, j), \text{nat-to-int}(x, i)))$

EVENT: Disable head-int-crock.

```
; unsigned interpretation of rem.
```

THEOREM: rem-nat
 $\text{nat-range}(x, n)$
 $\rightarrow (\text{nat-to-uint}(\text{rem}(n, x, y))$
 $\quad = \text{if } \text{nat-to-uint}(x) = 0 \text{ then } \text{nat-to-uint}(y) \bmod \exp(2, n)$
 $\quad \text{else } \text{nat-to-uint}(y) \bmod \text{nat-to-uint}(x) \text{ endif})$

```
; unsigned interpretation of quot.
```

THEOREM: quot-nat

nat-range $p(y, n)$

$\rightarrow$ (nat-to-uint(quot(n, x, y)) = (nat-to-uint(y) $\div$ nat-to-uint(x)))

EVENT: Disable int-range p .

; the only correct execution of DIV is when NO overflow occurs. We
; need to pass the guard in the specification. The following few
; lemmas are introduced for this purpose.

THEOREM: divu-overflow

divu-v(n, x, y)

= if uint-range p (nat-to-uint(y) $\div$ nat-to-uint(x), n) then 0
else 1 endif

THEOREM: divs-overflow

divs-v(n, x, i, y, j)

= if int-range p (iquotient(nat-to-int(y, j), nat-to-int(x, i)), n) then 0
else 1 endif

THEOREM: divs_3232-overflow

divs-v(32, x , 32, y , 32)

= if (nat-to-int($y, 32$) = -2147483648)
 $\wedge$ (nat-to-int($x, 32$) = -1) then 1
else 0 endif

EVENT: Disable divu-v.

EVENT: Disable divs-v.

; UNSIGNED INTERPRETATION OF ADDITION AND SUBTRACTION
;

THEOREM: add-nat-la

(nat-range $p(x, n) \wedge$ nat-range $p(y, n)$)

$\rightarrow$ (add(n, x, y)
= if ($x + y$) < exp(2, n) then $x + y$
else ($x + y$) - exp(2, n) endif)

THEOREM: sub-nat-la

(nat-range $p(x, n) \wedge$ nat-range $p(y, n)$)

$\rightarrow$ (sub(n, x, y)
= if $y < x$ then ($y +$ exp(2, n)) - x
else $y - x$ endif)

THEOREM: adder-nat-la

$$\begin{aligned}
& (\text{nat-range}(x, n) \wedge \text{nat-range}(y, n) \wedge \text{bitp}(c)) \\
\rightarrow & \text{adder}(n, c, x, y) \\
= & \text{if } (c + x + y) < \exp(2, n) \text{ then } c + x + y \\
& \text{else } (c + x + y) - \exp(2, n) \text{ endif}
\end{aligned}$$

THEOREM: subtracter-nat-la

$$\begin{aligned}
& (\text{nat-range}(x, n) \wedge \text{nat-range}(y, n) \wedge \text{bitp}(c)) \\
\rightarrow & \text{subtracter}(n, c, x, y) \\
= & \text{if } y < (c + x) \text{ then } (y + \exp(2, n)) - (c + x) \\
& \text{else } y - (c + x) \text{ endif}
\end{aligned}$$

THEOREM: plus-numberp

$$(y \notin \mathbf{N}) \rightarrow ((x + y) = \text{fix}(x))$$

; unsigned interpretation of adder.

THEOREM: adder-uint

$$\begin{aligned}
& (\text{nat-range}(x, n) \wedge \text{nat-range}(y, n) \wedge \text{bitp}(c)) \\
\rightarrow & (\text{nat-to-uint}(\text{adder}(n, c, x, y))) \\
= & \text{if } (c + \text{nat-to-uint}(x) + \text{nat-to-uint}(y)) < \exp(2, n) \\
& \text{then } c + \text{nat-to-uint}(x) + \text{nat-to-uint}(y) \\
& \text{else } (c + \text{nat-to-uint}(x) + \text{nat-to-uint}(y)) \\
& \quad - \exp(2, n) \text{ endif}
\end{aligned}$$

; unsigned interpretation of subtracter.

THEOREM: subtracter-uint

$$\begin{aligned}
& (\text{nat-range}(x, n) \wedge \text{nat-range}(y, n) \wedge \text{bitp}(c)) \\
\rightarrow & (\text{nat-to-uint}(\text{subtracter}(n, c, x, y))) \\
= & \text{if } \text{nat-to-uint}(y) < (c + \text{nat-to-uint}(x)) \\
& \text{then } (\text{nat-to-uint}(y) + \exp(2, n)) \\
& \quad - (c + \text{nat-to-uint}(x)) \\
& \text{else } \text{nat-to-uint}(y) - (c + \text{nat-to-uint}(x)) \text{ endif}
\end{aligned}$$

; unsigned interpretation of add.

THEOREM: add-uint

$$\begin{aligned}
& (\text{nat-range}(x, n) \wedge \text{nat-range}(y, n)) \\
\rightarrow & (\text{nat-to-uint}(\text{add}(n, x, y))) \\
= & \text{if } (\text{nat-to-uint}(x) + \text{nat-to-uint}(y)) < \exp(2, n) \\
& \text{then } \text{nat-to-uint}(x) + \text{nat-to-uint}(y) \\
& \text{else } (\text{nat-to-uint}(x) + \text{nat-to-uint}(y)) - \exp(2, n) \text{ endif}
\end{aligned}$$

; unsigned interpretation of sub.

THEOREM: sub-uint
 $(\text{nat-range}(x, n) \wedge \text{nat-range}(y, n))$
 $\rightarrow (\text{nat-to-uint}(\text{add}(n, x, \text{neg}(n, y))))$
 $= \text{if } \text{nat-to-uint}(x) < \text{nat-to-uint}(y)$
 $\quad \text{then } (\text{nat-to-uint}(x) + \text{exp}(2, n)) - \text{nat-to-uint}(y)$
 $\quad \text{else } \text{nat-to-uint}(x) - \text{nat-to-uint}(y) \text{ endif}$

EVENT: Disable plus-numberp.

EVENT: Disable add-nat-la.

EVENT: Disable sub-nat-la.

EVENT: Disable adder-nat-la.

EVENT: Disable subtracter-nat-la.

; THEOREMS OF ADDER
 ;

THEOREM: adder-shift-carry
 $\text{add}(n, c, x, \text{add}(n, d, y, z)) = \text{add}(n, d, x, \text{add}(n, c, y, z))$

THEOREM: adder-commutativity
 $\text{add}(n, c, x, y) = \text{add}(n, c, y, x)$

THEOREM: adder-associativity
 $\text{add}(n, d, \text{add}(n, c, x, y), z) = \text{add}(n, c, x, \text{add}(n, d, y, z))$

; another commutativity theorem about adder.

THEOREM: adder-commutativity1
 $\text{add}(n, c, x, \text{add}(n, d, y, z)) = \text{add}(n, c, y, \text{add}(n, d, x, z))$

; THEOREMS ABOUT ADD AND SUB
 ;
 ; trivial!

THEOREM: add-of-0
 $\text{add}(0, x, y) = 0$

; $x + 0 = x$!

THEOREM: add-0
 $(\text{add}(n, x, 0) = \text{head}(x, n)) \wedge (\text{add}(n, 0, x) = \text{head}(x, n))$

; $x - 0 = x!$

THEOREM: sub-0
 $\text{sub}(n, 0, x) = \text{head}(x, n)$

; $x - x = 0!$

THEOREM: sub-x-x
 $\text{sub}(n, x, x) = 0$

; commutativity of addition : $x + y = y + x$.

THEOREM: add-commutativity
 $\text{add}(n, x, y) = \text{add}(n, y, x)$

; associativity of addition: $(x + y) + z = x + (y + z)$.

THEOREM: add-associativity
 $\text{add}(n, \text{add}(n, x, y), z) = \text{add}(n, x, \text{add}(n, y, z))$

; another commutativity of addition: $x + (y + z) = y + (x + z)$.

THEOREM: add-commutativity1
 $\text{add}(n, x, \text{add}(n, y, z)) = \text{add}(n, y, \text{add}(n, x, z))$

THEOREM: remainder-leq
 $x \not\leq (x \bmod y)$

THEOREM: add-leq
 $(x + y) \not\leq \text{add}(n, x, y)$

THEOREM: sub-leq-la
 $(x \leq y) \rightarrow ((y - x) \not\leq \text{sub}(n, x, y))$

EVENT: Disable remainder-leq.

; addition and subtraction are always in "good" range.

THEOREM: adder-nat-rangep
 $\text{nat-rangep}(\text{adder}(n, c, x, y), n)$

THEOREM: subtracter-nat-rangep
 $\text{nat-rangep}(\text{subtracter}(n, c, y, x), n)$

THEOREM: add-nat-range
 $\text{nat-range}(\text{add}(n, x, y), n)$

THEOREM: sub-nat-range
 $\text{nat-range}(\text{sub}(n, x, y), n)$

THEOREM: adder-head
 $(\text{adder}(n, c, \text{head}(x, n), y) = \text{adder}(n, c, x, y))$
 $\wedge (\text{adder}(n, c, x, \text{head}(y, n)) = \text{adder}(n, c, x, y))$

THEOREM: add-head
 $(\text{add}(n, \text{head}(x, n), y) = \text{add}(n, x, y)) \wedge (\text{add}(n, x, \text{head}(y, n)) = \text{add}(n, x, y))$

; $x + y = x - z \iff y + z = 0$.

THEOREM: add-sub-cancel
 $(\text{nat-range}(y, n) \wedge \text{nat-range}(z, n))$
 $\rightarrow ((\text{add}(n, x, y) = \text{sub}(n, z, x)) = (\text{add}(n, y, z) = 0))$

; $x - y = 0 \iff x = y$.

THEOREM: sub-equal-0
 $(\text{nat-range}(x, n) \wedge \text{nat-range}(y, n))$
 $\rightarrow ((\text{sub}(n, x, y) = 0) = (\text{fix}(x) = \text{fix}(y)))$

; a bridge from add to adder.

THEOREM: add-adder
 $\text{add}(n, x, y) = \text{adder}(n, 0, x, y)$

THEOREM: plus-add1-sub1
 $(y \neq 0) \rightarrow ((1 + (x + (y - 1))) = (x + y))$

; a bridge from sub to adder.

THEOREM: sub-adder
 $\text{sub}(n, y, x) = \text{adder}(n, 1, x, \text{lognot}(n, y))$

THEOREM: add-neg-adder
 $\text{add}(n, x, \text{neg}(n, y)) = \text{adder}(n, 1, x, \text{lognot}(n, y))$

; plus-add1-sub1 is a very ‘‘dangerous’’ event. To use it, you need
 ; to enable it first.

EVENT: Disable plus-add1-sub1.

EVENT: Disable add.

EVENT: Disable sub.

; $(x + y) - z = x + (y - z)$.

THEOREM: add-sub

$\text{sub}(n, z, \text{add}(n, x, y)) = \text{add}(n, x, \text{sub}(n, z, y))$

; $(x - y) + z = x + (z - y)$.

THEOREM: sub-add

$\text{add}(n, \text{sub}(n, y, x), z) = \text{add}(n, x, \text{sub}(n, y, z))$

; $(x - y) - z = x - (y + z)$.

THEOREM: sub-sub

$\text{sub}(n, z, \text{sub}(n, y, x)) = \text{sub}(n, \text{add}(n, y, z), x)$

; $x - (y - z) = x + (z - y)$.

THEOREM: sub-sub1

$\text{sub}(n, \text{sub}(n, z, y), x) = \text{add}(n, x, \text{sub}(n, y, z))$

; another commutativity of addition: $x + (y - z) = y + (x - z)$.

THEOREM: add-commutativity2

$\text{add}(n, x, \text{sub}(n, z, y)) = \text{add}(n, y, \text{sub}(n, z, x))$

; $x + y = x \iff y = 0$.

THEOREM: add-cancel0

$(\text{add}(n, x, y) = x) = ((x \in \mathbf{N}) \wedge \text{nat-range}(x, n) \wedge (\text{head}(y, n) = 0))$

; $x - y = x \iff y = 0$.

THEOREM: sub-cancel0

$(\text{nat-range}(x, n) \wedge \text{nat-range}(y, n))$
 $\rightarrow ((\text{sub}(n, y, x) = x) = ((x \in \mathbf{N}) \wedge (y \simeq 0)))$

THEOREM: adder-cancel

$(\text{adder}(n, c, x, y) = \text{adder}(n, c, x, z)) = (\text{head}(y, n) = \text{head}(z, n))$

; $x + y = x + z \iff y = z$.

THEOREM: add-cancel

$(\text{add}(n, x, y) = \text{add}(n, x, z)) = (\text{head}(y, n) = \text{head}(z, n))$

; $x - y = x - z \iff y = z$.

THEOREM: sub-cancel

$(\text{nat-range}(y, n) \wedge \text{nat-range}(z, n))$
 $\rightarrow ((\text{sub}(n, y, x) = \text{sub}(n, z, x)) = (\text{fix}(y) = \text{fix}(z)))$

EVENT: Disable add-adder.

EVENT: Disable sub-adder.

; $x + (y - y) = x!$

THEOREM: addy-y

$\text{nat-range}(x, n) \rightarrow (\text{add}(n, x, \text{sub}(n, y, y)) = \text{fix}(x))$

; $y + (x - y) = x!$

THEOREM: addy-y1

$\text{nat-range}(x, n) \rightarrow (\text{add}(n, y, \text{sub}(n, y, x)) = \text{fix}(x))$

; INTEGER INTERPRETATION OF ADDITION AND SUBTRACTION

; a bridge for adder-int.

THEOREM: adder-int-bridge

$(\text{nat-range}(x, n) \wedge \text{nat-range}(y, n) \wedge \text{bitp}(c))$

$\rightarrow (\text{nat-to-int}(\text{adder}(n, c, x, y), n)$
 $= \text{adder-int-end}(n, c, \text{nat-to-int}(x, n), \text{nat-to-int}(y, n)))$

EVENT: Disable adder-int-end.

EVENT: Disable adder.

; integer interpretation of adder.

THEOREM: adder-int

$(\text{nat-range}(x, n) \wedge \text{nat-range}(y, n) \wedge \text{bitp}(c))$

$\rightarrow (\text{nat-to-int}(\text{adder}(n, c, x, y), n)$
 $= \text{if int-range}(\text{iplus}(\text{nat-to-int}(x, n), \text{iplus}(\text{nat-to-int}(y, n), c)),$
 $n)$
 then $\text{iplus}(\text{nat-to-int}(x, n), \text{iplus}(\text{nat-to-int}(y, n), c))$
 elseif $\text{negativep}(\text{nat-to-int}(x, n))$
 then $\text{iplus}(\text{nat-to-int}(x, n),$
 $\text{iplus}(\text{nat-to-int}(y, n), \text{iplus}(c, \text{exp}(2, n))))$
 else $\text{iplus}(\text{nat-to-int}(x, n),$
 $\text{iplus}(\text{nat-to-int}(y, n), \text{iplus}(c, -\text{exp}(2, n)))) \text{ endif})$

; integer interpretation of subtracter.

THEOREM: subtracter-int

```

(nat-range (x, n) ∧ nat-range (y, n) ∧ bitp (c) ∧ (n ≠ 0))
→ (nat-to-int (subtracter (n, c, x, y), n)
   =  if int-range (idifference (nat-to-int (y, n),
                                iplus (nat-to-int (x, n), c)),
                                n)
      then idifference (nat-to-int (y, n), iplus (nat-to-int (x, n), c))
      elseif negativep (nat-to-int (y, n))
      then idifference (iplus (nat-to-int (y, n), exp (2, n)),
                        iplus (nat-to-int (x, n), c))
      else idifference (iplus (nat-to-int (y, n), - exp (2, n)),
                        iplus (nat-to-int (x, n), c)) endif)

```

EVENT: Disable subtracter.

; integer interpretation of add.

THEOREM: add-int

```

(nat-range (x, n) ∧ nat-range (y, n))
→ (nat-to-int (add (n, x, y), n)
   =  if int-range (iplus (nat-to-int (x, n), nat-to-int (y, n)), n)
      then iplus (nat-to-int (x, n), nat-to-int (y, n))
      elseif negativep (nat-to-int (x, n))
      then iplus (nat-to-int (x, n), iplus (nat-to-int (y, n), exp (2, n)))
      else iplus (nat-to-int (x, n),
                  iplus (nat-to-int (y, n), - exp (2, n))) endif)

```

; integer interpretation of sub.

THEOREM: sub-int

```

(nat-range (x, n) ∧ nat-range (y, n) ∧ (n ≠ 0))
→ (nat-to-int (add (n, x, neg (n, y)), n)
   =  if int-range (idifference (nat-to-int (x, n), nat-to-int (y, n)), n)
      then idifference (nat-to-int (x, n), nat-to-int (y, n))
      elseif negativep (nat-to-int (x, n))
      then idifference (iplus (nat-to-int (x, n), exp (2, n)),
                        nat-to-int (y, n))
      else idifference (iplus (nat-to-int (x, n), - exp (2, n)),
                        nat-to-int (y, n)) endif)

```

EVENT: Disable add-neg-adder.

```

;          THEOREMS ABOUT HEAD, TAIL, APP, AND REPLACE
;

```

THEOREM: head-head

$$\begin{aligned} & \text{head}(\text{head}(x, m), n) \\ &= \text{if } n < m \text{ then head}(x, n) \\ & \quad \text{else head}(x, m) \text{ endif} \end{aligned}$$

THEOREM: tail-tail

$$\text{tail}(\text{tail}(x, m), n) = \text{tail}(x, m + n)$$

THEOREM: head-plus-cancel0

$$(\text{head}(i + j, n) = \text{head}(i, n)) = (\text{head}(j, n) = 0)$$

THEOREM: head-plus-cancel

$$(\text{head}(x + y, n) = \text{head}(x + z, n)) = (\text{head}(y, n) = \text{head}(z, n))$$

THEOREM: head-plus-head

$$\text{head}(x + \text{head}(y, n), n) = \text{head}(x + y, n)$$

THEOREM: remainder-plus-times-exp2-1

$$\begin{aligned} & (j \leq i) \\ & \rightarrow (((x + (y * \exp(2, i))) \bmod \exp(2, j)) = (x \bmod \exp(2, j))) \end{aligned}$$

THEOREM: remainder2-plus-times-exp2

$$(i \neq 0) \rightarrow (((x + (y * \exp(2, i))) \bmod 2) = (x \bmod 2))$$

THEOREM: remainder-plus-times-exp2-2

$$\begin{aligned} & ((x < \exp(2, i)) \wedge (i \leq j)) \\ & \rightarrow (((x + (y * \exp(2, i))) \bmod \exp(2, j)) \\ & \quad = \text{if } i < j \text{ then } x + ((y \bmod \exp(2, j - i)) * \exp(2, i)) \\ & \quad \text{else fix}(x) \text{ endif}) \end{aligned}$$

THEOREM: quotient-plus-times-exp2-1

$$\begin{aligned} & (j < i) \\ & \rightarrow (((x + (y * \exp(2, i))) \div \exp(2, j)) \\ & \quad = ((x \div \exp(2, j)) + (y * \exp(2, i - j)))) \end{aligned}$$

THEOREM: exp2-leq

$$(i \leq j) \rightarrow (\exp(2, j) \not< \exp(2, i))$$

THEOREM: quotient-plus-times-exp2-2

$$\begin{aligned} & ((i \leq j) \wedge (x < \exp(2, i))) \\ & \rightarrow (((x + (y * \exp(2, i))) \div \exp(2, j)) = (y \div \exp(2, j - i))) \end{aligned}$$

THEOREM: remainder-quotient-exp2

$$\begin{aligned} & ((x \bmod \exp(2, i)) \div \exp(2, j)) \\ &= \text{if } j < i \text{ then } (x \div \exp(2, j)) \bmod \exp(2, i - j) \\ & \quad \text{else } 0 \text{ endif} \end{aligned}$$

EVENT: Disable exp2-leq.

THEOREM: head-replace
 $\text{head}(\text{replace}(m, x, y), n)$
 $= \text{if } m < n \text{ then } \text{replace}(m, x, \text{head}(y, n))$
 $\text{else } \text{head}(x, n) \text{ endif}$

THEOREM: tail-head
 $\text{tail}(\text{head}(x, m), n) = \text{head}(\text{tail}(x, n), m - n)$

THEOREM: tail-app
 $\text{tail}(\text{app}(i, x, y), j)$
 $= \text{if } j < i \text{ then } \text{app}(i - j, \text{tail}(x, j), y)$
 $\text{else } \text{tail}(y, j - i) \text{ endif}$

THEOREM: tail-replace
 $\text{tail}(\text{replace}(i, x, y), j)$
 $= \text{if } j < i \text{ then } \text{replace}(i - j, \text{tail}(x, j), \text{tail}(y, j))$
 $\text{else } \text{tail}(y, j) \text{ endif}$

THEOREM: replace-reflex
 $\text{replace}(n, x, x) = \text{fix}(x)$

THEOREM: replace-head
 $(n \leq m) \rightarrow (\text{replace}(n, \text{head}(x, m), x) = \text{fix}(x))$

THEOREM: replace-associativity
 $\text{replace}(n, x, \text{replace}(n, y, z)) = \text{replace}(n, \text{replace}(n, x, y), z)$

THEOREM: replace-leq1
 $(j \leq i) \rightarrow (\text{replace}(i, x, \text{replace}(j, y, z)) = \text{replace}(i, x, z))$

THEOREM: replace-leq
 $(i \leq j) \rightarrow (\text{replace}(i, \text{replace}(j, x, y), z) = \text{replace}(i, x, z))$

THEOREM: head-app
 $(i \leq j) \rightarrow (\text{head}(\text{app}(j, x, y), i) = \text{head}(x, i))$

; a few basic lemmas about bcar and bcdr. They are useful in induction
; proofs with bcar and bcdr disabled.

THEOREM: bcar-nonnumberp
 $(x \notin \mathbf{N}) \rightarrow (\text{bcar}(x) = 0)$

THEOREM: bcdr-nonnumberp
 $(x \notin \mathbf{N}) \rightarrow (\text{bcdr}(x) = 0)$

THEOREM: bcar-1

$$(\text{bcar}(2 * x) = 0) \wedge (\text{bcar}(1 + (2 * x)) = 1)$$

THEOREM: bcdr-1

$$(\text{bcdr}(2 * x) = \text{fix}(x)) \wedge (\text{bcdr}(1 + (2 * x)) = \text{fix}(x))$$

THEOREM: bcar-2

$$\text{bcar}(x + (2 * y)) = \text{bcar}(x)$$

THEOREM: bcdr-2

$$\text{bcdr}(x + (2 * y)) = (\text{bcdr}(x) + y)$$

; the first "bit" of head.

THEOREM: bcar-head

$$\begin{aligned} & \text{bcar}(\text{head}(x, n)) \\ &= \text{if } n \simeq 0 \text{ then } 0 \\ & \quad \text{else } \text{bcar}(x) \text{ endif} \end{aligned}$$

; the first "bit" of app.

THEOREM: bcar-app

$$\begin{aligned} & \text{bcar}(\text{app}(n, x, y)) \\ &= \text{if } n \simeq 0 \text{ then } \text{bcar}(y) \\ & \quad \text{else } \text{bcar}(x) \text{ endif} \end{aligned}$$

; the first "bit" of replace.

THEOREM: bcar-replace

$$\begin{aligned} & \text{bcar}(\text{replace}(n, x, y)) \\ &= \text{if } n \simeq 0 \text{ then } \text{bcar}(y) \\ & \quad \text{else } \text{bcar}(x) \text{ endif} \end{aligned}$$

THEOREM: bcar-lessp

$$\text{bcar}(x) < 2$$

THEOREM: bcdr-lessp

$$(\text{bcdr}(x) < y) = (x < (2 * y))$$

THEOREM: app-associativity

$$\text{app}(n1, x, \text{app}(n2, y, z)) = \text{app}(n1 + n2, \text{app}(n1, x, y), z)$$

THEOREM: app-head-tail

$$\text{app}(n, \text{head}(x, n), \text{tail}(x, n)) = \text{fix}(x)$$

THEOREM: head-app-head-tail

$$\text{app}(m, \text{head}(x, m), \text{head}(\text{tail}(x, m), n)) = \text{head}(x, m + n)$$

EVENT: Disable head.

EVENT: Disable tail.

EVENT: Disable bcar.

EVENT: Disable bcdl.

THEOREM: app-nat-range
 $(m \leq n) \rightarrow (\text{nat-range}(\text{app}(m, x, y), n) = \text{nat-range}(y, n - m))$

THEOREM: replace-nat-range
 $(m \leq n) \rightarrow (\text{nat-range}(\text{replace}(m, x, y), n) = \text{nat-range}(y, n))$

EVENT: Disable app.

EVENT: Disable replace.

;
BITN

THEOREM: bitn-0
 $(\text{bitn}(0, n) = 0) \wedge ((x \notin \mathbf{N}) \rightarrow (\text{bitn}(x, i) = 0))$

THEOREM: bitn-0-1
 $\text{bitn}(x, n) < 2$

THEOREM: bitn-head
 $\text{bitn}(\text{head}(x, i), j)$
 $= \text{if } j < i \text{ then } \text{bitn}(x, j)$
 $\text{else } 0 \text{ endif}$

THEOREM: bitn-tail
 $\text{bitn}(\text{tail}(x, i), j) = \text{bitn}(x, i + j)$

THEOREM: bitn-app
 $\text{bitn}(\text{app}(n, x, y), k)$
 $= \text{if } k < n \text{ then } \text{bitn}(x, k)$
 $\text{else } \text{bitn}(y, k - n) \text{ endif}$

THEOREM: bitn-replace
 $\text{bitn}(\text{replace}(n, x, y), k)$
 $= \text{if } k < n \text{ then } \text{bitn}(x, k)$
 $\text{else } \text{bitn}(y, k) \text{ endif}$

EVENT: Disable bitn.

THEOREM: bitn-lognot

nat-rangep (x, n)
 $\rightarrow$ (bitn (lognot (n, x) , i)
 $=$ **if** $i < n$ **then** b-not (bitn (x, i))
else 0 endif)

;
LOGAND, LOGOR, AND LOGEOR.

THEOREM: logor-equal-0

(logor $(x, y) = 0$) = $((x \simeq 0) \wedge (y \simeq 0))$

THEOREM: logeor-equal-0

(logeor $(x, y) = 0$) = $(\text{fix}(x) = \text{fix}(y))$

THEOREM: logand-commutativity

logand $(x, y) = \text{logand}(y, x)$

THEOREM: logor-commutativity

logor $(x, y) = \text{logor}(y, x)$

THEOREM: logeor-commutativity

logeor $(x, y) = \text{logeor}(y, x)$

THEOREM: logand-logor

logand $(x, \text{logor}(y, z)) = \text{logor}(\text{logand}(x, y), \text{logand}(x, z))$

THEOREM: logand-logeor

logand $(x, \text{logeor}(y, z)) = \text{logeor}(\text{logand}(x, y), \text{logand}(x, z))$

THEOREM: logand-uint-la

nat-to-uint (logand $(\exp(2, i) - 1, y)$) = $(\text{nat-to-uint}(y) \bmod \exp(2, i))$

THEOREM: logand-uint

$(\exp(2, \log(2, 1 + x)) = (1 + x))$
 $\rightarrow$ $(\text{nat-to-uint}(\text{logand}(x, y)) = (\text{nat-to-uint}(y) \bmod (1 + x)))$

;
HEAD WITH READ/WRITE AND READM/WITEM

THEOREM: head-readp

$(n \leq k) \rightarrow (\text{readp}(\text{head}(x, k), n, \text{map}) = \text{readp}(x, n, \text{map}))$

THEOREM: head-pc-readp

$(n \leq k) \rightarrow (\text{pc-readp}(\text{head}(x, k), n, \text{map}) = \text{pc-readp}(x, n, \text{map}))$

THEOREM: head-writep

$$(n \leq k) \rightarrow (\text{writep}(\text{head}(x, k), n, \text{map}) = \text{writep}(x, n, \text{map}))$$

THEOREM: head-read

$$(n \leq k) \rightarrow (\text{read}(\text{head}(x, k), n, \text{bt}) = \text{read}(x, n, \text{bt}))$$

THEOREM: head-pc-read

$$(n \leq k) \rightarrow (\text{pc-read}(\text{head}(x, k), n, \text{bt}) = \text{pc-read}(x, n, \text{bt}))$$

THEOREM: head-write

$$(n \leq k) \rightarrow (\text{write}(\text{value}, \text{head}(x, k), n, \text{bt}) = \text{write}(\text{value}, x, n, \text{bt}))$$

THEOREM: head-byte-readp

$$\text{byte-readp}(\text{head}(x, 32), \text{mem}) = \text{byte-readp}(x, \text{mem})$$

THEOREM: head-pc-byte-readp

$$\text{pc-byte-readp}(\text{head}(x, 32), \text{mem}) = \text{pc-byte-readp}(x, \text{mem})$$

THEOREM: head-byte-writep

$$\text{byte-writep}(\text{head}(x, 32), \text{mem}) = \text{byte-writep}(x, \text{mem})$$

THEOREM: head-byte-read

$$\text{byte-read}(\text{head}(x, 32), \text{mem}) = \text{byte-read}(x, \text{mem})$$

THEOREM: head-byte-pc-read

$$\text{pc-byte-read}(\text{head}(x, 32), \text{mem}) = \text{pc-byte-read}(x, \text{mem})$$

THEOREM: head-byte-write

$$\text{byte-write}(\text{value}, \text{head}(x, 32), \text{mem}) = \text{byte-write}(\text{value}, x, \text{mem})$$

THEOREM: head-read-memp

$$\text{read-memp}(\text{head}(x, 32), \text{mem}, k) = \text{read-memp}(x, \text{mem}, k)$$

THEOREM: head-pc-read-memp

$$\text{pc-read-memp}(\text{head}(x, 32), \text{mem}, k) = \text{pc-read-memp}(x, \text{mem}, k)$$

THEOREM: head-write-memp

$$\text{write-memp}(\text{head}(x, 32), \text{mem}, k) = \text{write-memp}(x, \text{mem}, k)$$

THEOREM: head-read-mem

$$\text{read-mem}(\text{head}(x, 32), \text{mem}, k) = \text{read-mem}(x, \text{mem}, k)$$

THEOREM: head-pc-read-mem

$$\text{pc-read-mem}(\text{head}(x, 32), \text{mem}, k) = \text{pc-read-mem}(x, \text{mem}, k)$$

THEOREM: head-write-mem

$$\text{write-mem}(\text{value}, \text{head}(x, 32), \text{mem}, k) = \text{write-mem}(\text{value}, x, \text{mem}, k)$$

THEOREM: head-readm-mem
 $\text{readm-mem}(\text{opsz}, \text{head}(x, 32), \text{mem}, n) = \text{readm-mem}(\text{opsz}, x, \text{mem}, n)$

;

NAT-RANGE

THEOREM: nat-range-of-0
 $\text{nat-range}(0, n)$

THEOREM: nat-range-ub
 $\text{nat-range}(\exp(2, k) - 1, n) = (n \not\leq k)$

THEOREM: nat-range-0
 $\text{nat-range}(x, 0) = (x \simeq 0)$

THEOREM: nat-plus-range
 $\text{nat-range}(x, k) \rightarrow (\text{nat-range}(x, k + j) \wedge \text{nat-range}(x, j + k))$

THEOREM: sub1-nat-range
 $\text{nat-range}(x, n) \rightarrow \text{nat-range}(x - 1, n)$

THEOREM: sub1-times2-nat-range
 $\text{nat-range}((2 * x) - 1, 1 + n) = \text{nat-range}(x - 1, n)$

THEOREM: difference-nat-range
 $\text{nat-range}(x, n) \rightarrow \text{nat-range}(x - y, n)$

THEOREM: quotient-nat-range
 $\text{nat-range}(x, n) \rightarrow \text{nat-range}(x \div y, n)$

THEOREM: times-exp2-nat-range
 $\text{nat-range}(x * \exp(2, i), n) = \text{nat-range}(x, n - i)$

THEOREM: logand-nat-range
 $(\text{nat-range}(x, n) \vee \text{nat-range}(y, n)) \rightarrow \text{nat-range}(\text{logand}(x, y), n)$

THEOREM: logor-nat-range
 $\text{nat-range}(\text{logor}(x, y), n) = (\text{nat-range}(x, n) \wedge \text{nat-range}(y, n))$

THEOREM: logeor-nat-range
 $(\text{nat-range}(x, n) \wedge \text{nat-range}(y, n)) \rightarrow \text{nat-range}(\text{logeor}(x, y), n)$

THEOREM: head-nat-range
 $(m \leq n) \rightarrow \text{nat-range}(\text{head}(x, m), n)$

THEOREM: tail-nat-range
 $\text{nat-range}(\text{tail}(x, m), n) = \text{nat-range}(x, m + n)$

THEOREM: read-rn-nat-range
 $\text{nat-range}(\text{read-rn}(n, rn, rfile), n)$

THEOREM: byte-read-nat-range
 $(8 \leq n) \rightarrow \text{nat-range}(\text{byte-read}(x, mem), n)$

THEOREM: pc-byte-read-nat-range
 $\text{nat-range}(\text{pc-byte-read}(x, mem), 8)$

EVENT: Disable nat-range.

THEOREM: mulu-nat-range
 $\text{nat-range}(\text{mulu}(n, x, y, i), n)$

THEOREM: muls-nat-range
 $\text{nat-range}(\text{muls}(n, x, y, i), n)$

THEOREM: quot-nat-range
 $\text{nat-range}(\text{quot}(n, x, y), n)$

THEOREM: rem-nat-range
 $\text{nat-range}(\text{rem}(n, x, y), n)$

THEOREM: iquot-nat-range
 $\text{nat-range}(\text{iquot}(n, x, i, y, j), n)$

THEOREM: irem-nat-range
 $\text{nat-range}(\text{irem}(n, x, i, y, j), n)$

THEOREM: neg-nat-range
 $\text{nat-range}(\text{neg}(n, x), n)$

; EXT

THEOREM: ext-0
 $\text{ext}(n, 0, size) = 0$

THEOREM: ext-lemma
 $(size \leq n) \rightarrow (\text{ext}(n, x, size) = \text{head}(x, size))$

THEOREM: head-ext
 $((i \leq j) \wedge (j \leq k)) \rightarrow (\text{head}(\text{ext}(j, x, k), i) = \text{head}(x, i))$

THEOREM: ext-nat-range
 $\text{nat-range}(\text{ext}(n, x, size), size)$

EVENT: Disable ext.

; CVZNX-FLAGS
 THEOREM: set-cvznx-c
 $\text{ccr-c}(\text{set-cvznx}(\text{cvznx}(c, v, z, n, x), \text{ccr})) = \text{fix-bit}(c)$
 THEOREM: set-cvznx-v
 $\text{ccr-v}(\text{set-cvznx}(\text{cvznx}(c, v, z, n, x), \text{ccr})) = \text{fix-bit}(v)$
 THEOREM: set-cvznx-z
 $\text{ccr-z}(\text{set-cvznx}(\text{cvznx}(c, v, z, n, x), \text{ccr})) = \text{fix-bit}(z)$
 THEOREM: set-cvznx-n
 $\text{ccr-n}(\text{set-cvznx}(\text{cvznx}(c, v, z, n, x), \text{ccr})) = \text{fix-bit}(n)$
 THEOREM: set-cvznx-x
 $\text{ccr-x}(\text{set-cvznx}(\text{cvznx}(c, v, z, n, x), \text{ccr})) = \text{fix-bit}(x)$
 ; the new cvznx-flags replaces the old ones.
 THEOREM: set-set-cvznx1
 $\text{set-cvznx}(x, \text{set-cvznx}(y, \text{ccr})) = \text{set-cvznx}(x, \text{ccr})$
 THEOREM: set-cvznx-ccr
 $\text{set-cvznx}(\text{ccr}, \text{ccr}) = \text{fix}(\text{ccr})$
 THEOREM: set-set-cvznx2
 $\text{set-cvznx}(\text{set-cvznx}(x, \text{ccr}), \text{ccr}) = \text{set-cvznx}(x, \text{ccr})$
 THEOREM: set-cvznx-nat-range
 $\text{nat-range}(\text{set-cvznx}(x, \text{ccr}), 8) = \text{nat-range}(\text{ccr}, 8)$
 EVENT: Disable ccr-c.

 EVENT: Disable ccr-v.

 EVENT: Disable ccr-z.

 EVENT: Disable ccr-n.

 EVENT: Disable ccr-x.

 EVENT: Disable cvznx.

 EVENT: Disable set-cvznx.

```

;                                     MC ACCESSORS

THEOREM: mc-status-rewrite
mc-status (mc-state (status, rfile, pc, ccr, mem)) = status

THEOREM: mc-rfile-rewrite
mc-rfile (mc-state (status, rfile, pc, ccr, mem)) = rfile

THEOREM: mc-pc-rewrite
nat-range (pc, 32)
→ (mc-pc (mc-state (status, rfile, pc, ccr, mem)) = fix (pc))

THEOREM: mc-ccr-rewrite
nat-range (ccr, 8)
→ (mc-ccr (mc-state (status, rfile, pc, ccr, mem)) = fix (ccr))

THEOREM: mc-mem-rewrite
mc-mem (mc-state (status, rfile, pc, ccr, mem)) = mem

; lemmas about those mc accessors.

THEOREM: mc-pc-range
nat-range (mc-pc (s), 32)

THEOREM: mc-ccr-range
nat-range (mc-ccr (s), 8)

EVENT: Disable mc-status.

EVENT: Disable mc-rfile.

EVENT: Disable mc-pc.

EVENT: Disable mc-ccr.

EVENT: Disable mc-mem.

EVENT: Disable mc-state.

```

```

;                                     THE BASIC GET-NTH/PUT-NTH RELATIONS
;

```

```

THEOREM: get-nth-0
( $i \notin \mathbf{N}$ ) → (get-nth ( $i$ ,  $lst$ ) = get-nth (0,  $lst$ ))

```

THEOREM: put-nth-0
 $(i \notin \mathbf{N}) \rightarrow (\text{put-nth}(v, i, lst) = \text{put-nth}(v, 0, lst))$

EVENT: Disable get-nth-0.

EVENT: Disable put-nth-0.

THEOREM: get-nth-nil
 $(\text{get-nth}(n, \mathbf{nil}) = 0) \wedge (\text{get-nth}(n, 0) = 0)$

THEOREM: get-put
 $\text{get-nth}(n, \text{put-nth}(value, m, lst))$
 $= \text{if } \text{fix}(m) = \text{fix}(n) \text{ then } value$
 $\quad \text{else } \text{get-nth}(n, lst) \text{ endif}$

THEOREM: put-put
 $\text{put-nth}(value1, n, \text{put-nth}(value, n, lst)) = \text{put-nth}(value1, n, lst)$

THEOREM: put-get
 $(n < \text{len}(lst)) \rightarrow (\text{put-nth}(\text{get-nth}(n, lst), n, lst) = lst)$

THEOREM: put-nth-len
 $\text{len}(\text{put-nth}(value, n, lst))$
 $= \text{if } n < \text{len}(lst) \text{ then } \text{len}(lst)$
 $\quad \text{else } 1 + n \text{ endif}$

; THE BASIC READ-RN/WRITE-RN RELATIONS
 ;

THEOREM: head-read-rn
 $(n1 \leq n2) \rightarrow (\text{head}(\text{read-rn}(n2, rn, rfile), n1) = \text{read-rn}(n1, rn, rfile))$

; stop proving.

DEFINITION:
 $\text{read-write-rn-end}(n2, rn, n1, value, rm, rfile)$
 $= \text{read-rn}(n2, rn, \text{write-rn}(n1, value, rm, rfile))$

THEOREM: read-write-rn
 $\text{read-rn}(n2, rn, \text{write-rn}(n1, value, rm, rfile))$
 $= \text{if } \text{fix}(rm) = \text{fix}(rn)$
 $\quad \text{then if } n2 \leq n1 \text{ then } \text{head}(value, n2)$
 $\quad \quad \text{else } \text{replace}(n1, value, \text{read-rn}(n2, rn, rfile)) \text{ endif}$
 $\quad \text{else } \text{read-rn}(n2, rn, rfile) \text{ endif}$

THEOREM: write-write-rn
 $(n1 \leq n2)$
 $\rightarrow$ (write-rn ($n2$, $v2$, rn , write-rn ($n1$, $v1$, rn , $rfile$)))
 $=$ write-rn ($n2$, $v2$, rn , $rfile$)

THEOREM: write-rn-len
len(write-rn ($oplen$, $value$, rn , $rfile$))
 $=$ **if** $rn < \text{len}(rfile)$ **then** len($rfile$)
else $1 + rn$ **endif**

EVENT: Disable read-rn.

EVENT: Disable write-rn.

;
THE BASIC READM-RN/WRITEM-RN EVENTS
;

THEOREM: readm-rn-len
len(readm-rn ($oplen$, $rnlst$, $rfile$)) = len($rnlst$)

DEFINITION:
get-vlst ($oplen$, $value$, rn , $rnlst$, $vlst$)
 $=$ **if** $rnlst \simeq \text{nil}$ **then** $value$
elseif fix(rn) = fix(car($rnlst$))
then get-vlst ($oplen$, head(car($vlst$), $oplen$), rn , cdr($rnlst$), cdr($vlst$))
else get-vlst ($oplen$, $value$, rn , cdr($rnlst$), cdr($vlst$)) **endif**

; we deliberately put on the hypothesis (numberp $value$) to restrict the use
; of this lemma. The dead loop situation is thereby avoided.

THEOREM: get-vlst-member
($value \in \mathbf{N}$)
 $\rightarrow$ (get-vlst ($oplen$, $value$, rn , $rnlst$, $vlst$)
 $=$ **if** n-member(rn , $rnlst$) **then** get-vlst ($oplen$, **nil**, rn , $rnlst$, $vlst$)
else $value$ **endif**)

; stop proving.

DEFINITION:
read-writem-rn-end ($n1$, rn , $n2$, $vlst$, $rnlst$, $rfile$)
 $=$ read-rn ($n1$, rn , writem-rn ($n2$, $vlst$, $rnlst$, $rfile$))

THEOREM: read-writem-rn
read-rn ($n1$, rn , writem-rn ($n2$, $vlst$, $rnlst$, $rfile$))
 $=$ **if** n-member(rn , $rnlst$)

```

then if ( $n1 \leq n2$ )  $\wedge$  ( $n2 \leq 32$ )
  then get-vlst( $n1, 0, rn, rnlst, vlst$ )
  else read-witem-rn-end( $n1, rn, n2, vlst, rnlst, rfile$ ) endif
else read-rn( $n1, rn, rfile$ ) endif

```

EVENT: Disable read-witem-rn-end.

THEOREM: readm-write-rn
 $(\neg \text{n-member}(rn, rnlst))$
 $\rightarrow (\text{readm-rn}(n1, rnlst, \text{write-rn}(n2, value, rn, rfile))$
 $= \text{readm-rn}(n1, rnlst, rfile))$

THEOREM: read-rn-0
 $(rn \notin \mathbf{N}) \rightarrow (\text{read-rn}(oplen, rn, rfile) = \text{read-rn}(oplen, 0, rfile))$

THEOREM: get-vlst-readm-rn
 $(n1 \leq n2)$
 $\rightarrow (\text{get-vlst}(n1, 0, rn, rnlst, \text{readm-rn}(n2, rnlst, rfile))$
 $= \text{if n-member}(rn, rnlst) \text{ then read-rn}(n1, rn, rfile)$
 $\text{else } 0 \text{ endif})$

EVENT: Disable read-rn-0.

EVENT: Disable get-vlst-member.

EVENT: Disable get-vlst.

```

;          THE BASIC READ-MEM/WRITE-MEM EVENTS
;

```

THEOREM: pc-readp->readp
 $\text{pc-readp}(x, n, map) \rightarrow \text{readp}(x, n, map)$

THEOREM: writep->readp
 $\text{writep}(x, n, map) \rightarrow \text{readp}(x, n, map)$

```

; a read right after a write at the same location returns the new value
; just written. But a read right after a write at a different location
; has the same effect as a read made on the original binary tree.

```

DEFINITION:
 $\text{modn-eq}(n, x, y)$
 $= \text{if } n \simeq 0 \text{ then } t$
 $\text{else } (\text{bitn}(x, n-1) = \text{bitn}(y, n-1)) \wedge \text{modn-eq}(n-1, x, y) \text{ endif}$

THEOREM: read-write

$$\begin{aligned} & \text{read}(y, n, \text{write}(\text{value}, x, n, \text{bt})) \\ &= \text{if } \text{modn-eq}(n, x, y) \text{ then } \text{value} \\ & \quad \text{else } \text{read}(y, n, \text{bt}) \text{ endif} \end{aligned}$$

; pc should always be able to go through the memory.

THEOREM: pc-read-write

$$\begin{aligned} & (\text{writep}(x, n, \text{map}) \wedge \text{pc-readp}(y, n, \text{map})) \\ & \rightarrow (\text{pc-read}(y, n, \text{write}(\text{value}, x, n, \text{bt})) = \text{pc-read}(y, n, \text{bt})) \end{aligned}$$

; write twice at the same location is equivalent to the second write on
; the original binary tree.

THEOREM: write-write-la

$$\begin{aligned} & \text{write}(v2, y, n, \text{write}(v1, x, n, \text{bt})) \\ &= \text{if } \neg \text{modn-eq}(n, x, y) \text{ then } \text{write}(v1, x, n, \text{write}(v2, y, n, \text{bt})) \\ & \quad \text{else } \text{write}(v2, y, n, \text{bt}) \text{ endif} \end{aligned}$$

THEOREM: write-write

$$\text{write}(v2, x, n, \text{write}(v1, x, n, \text{bt})) = \text{write}(v2, x, n, \text{bt})$$

EVENT: Disable pc-read.

THEOREM: plus-times-equal

$$\begin{aligned} & ((a < k) \wedge (b < k)) \\ & \rightarrow (((a + (i * k)) = (b + (j * k))) \\ & \quad = ((\text{fix}(a) = \text{fix}(b)) \wedge (\text{fix}(i) = \text{fix}(j)))) \end{aligned}$$

THEOREM: app-cancel

$$\begin{aligned} & (\text{app}(n, x, y) = \text{app}(n, x1, y1)) \\ &= ((\text{head}(x, n) = \text{head}(x1, n)) \wedge (\text{fix}(y) = \text{fix}(y1))) \end{aligned}$$

EVENT: Disable plus-times-equal.

THEOREM: head-app-cancel

$$(\text{head}(x, n) = \text{app}(n, x1, y1)) = ((\text{head}(x, n) = \text{head}(x1, n)) \wedge (y1 \simeq 0))$$

THEOREM: head-recursion

$$\text{head}(x, 1 + n) = \text{app}(n, \text{head}(x, n), \text{bitn}(x, n))$$

THEOREM: modn-eq-equal

$$\text{modn-eq}(n, x, y) = (\text{head}(x, n) = \text{head}(y, n))$$

THEOREM: ext-equal
 $((n \leq size) \wedge (n \neq 0))$
 $\rightarrow ((\text{ext}(n, x, size) = \text{ext}(n, y, size)) = (\text{head}(x, n) = \text{head}(y, n)))$

THEOREM: ext-equal-0
 $((n \leq size) \wedge (n \neq 0))$
 $\rightarrow ((\text{ext}(n, x, size) = 0) = (\text{head}(x, n) = 0))$

EVENT: Disable head-recursion.

```
;                                BYTE-READ/BYTE-WRITE
;
;
; pc-byte-readp --> byte-readp.
```

THEOREM: pc-byte-readp->byte-readp
 $\text{pc-byte-readp}(x, mem) \rightarrow \text{byte-readp}(x, mem)$

```
; byte-writep --> byte-readp.
```

THEOREM: byte-writep->readp
 $\text{byte-writep}(x, mem) \rightarrow \text{byte-readp}(x, mem)$

THEOREM: byte-read-write
 $\text{byte-read}(x, \text{byte-write}(v, y, mem))$
 $=$ **if** mod32-eq(x, y)
 then if nat-range($v, 8$) **then** fix(v)
 else head($v, 8$) **endif**
 else byte-read(x, mem) **endif**

THEOREM: byte-write-write-la
 $\text{byte-write}(v2, y, \text{byte-write}(v1, x, mem))$
 $=$ **if** $\neg$ mod32-eq(x, y) **then** byte-write($v1, x, \text{byte-write}(v2, y, mem)$)
 else byte-write($v2, y, mem$) **endif**

THEOREM: byte-write-write
 $\text{byte-write}(v2, x, \text{byte-write}(v1, x, mem)) = \text{byte-write}(v2, x, mem)$

THEOREM: pc-byte-read-write
 $(\text{byte-writep}(x, mem) \wedge \text{pc-byte-readp}(y, mem))$
 $\rightarrow (\text{pc-byte-read}(y, \text{byte-write}(value, x, mem)) = \text{pc-byte-read}(y, mem))$

; write on memory does not change the properties of the memory.

THEOREM: byte-write-maintain-pc-byte-readp
 $\text{pc-byte-readp}(x, \text{byte-write}(value, y, mem)) = \text{pc-byte-readp}(x, mem)$

THEOREM: byte-write-maintain-byte-readp

$\text{byte-readp}(x, \text{byte-write}(value, y, mem)) = \text{byte-readp}(x, mem)$

THEOREM: byte-write-maintain-byte-writep

$\text{byte-writep}(x, \text{byte-write}(value, y, mem)) = \text{byte-writep}(x, mem)$

; a lemma useful to deal with read-mem/write-mem.

THEOREM: byte-write-app

$(8 \leq n) \rightarrow (\text{byte-write}(\text{app}(n, x, y), addr, mem) = \text{byte-write}(x, addr, mem))$

EVENT: Disable pc-byte-readp.

EVENT: Disable byte-readp.

EVENT: Disable byte-writep.

EVENT: Disable pc-byte-read.

EVENT: Disable byte-read.

EVENT: Disable byte-write.

; THE BASIC READ-MEM/WRITE-MEM EVENTS

DEFINITION:

$\text{mem-induct0}(k, n)$

= **if** $k \simeq 0$ **then** 0

else $\text{mem-induct0}(k - 1, n - 8)$ **endif**

THEOREM: pc-read-mem-nat-range

$(n = (8 * k)) \rightarrow \text{nat-range}(\text{pc-read-mem}(x, mem, k), n)$

THEOREM: read-mem-nat-range

$((8 * k) \leq n) \rightarrow \text{nat-range}(\text{read-mem}(x, mem, k), n)$

; write on memory does not change the properties of the memory. i.e. RAM is
; still RAM, ROM is still ROM, and UNAVAILABLE is still unavailable.

THEOREM: write-mem-maintain-pc-byte-readp

$\text{pc-byte-readp}(addr, \text{write-mem}(value, x, mem, k)) = \text{pc-byte-readp}(addr, mem)$

THEOREM: write-mem-maintain-byte-readp

$\text{byte-readp}(addr, \text{write-mem}(value, x, mem, k)) = \text{byte-readp}(addr, mem)$

THEOREM: write-mem-maintain-byte-writep
 $\text{byte-writep}(\text{addr}, \text{write-mem}(\text{value}, x, \text{mem}, k)) = \text{byte-writep}(\text{addr}, \text{mem})$

THEOREM: byte-write-maintain-pc-read-memp
 $\text{pc-read-memp}(\text{addr}, \text{byte-write}(\text{value}, x, \text{mem}), n) = \text{pc-read-memp}(\text{addr}, \text{mem}, n)$

THEOREM: byte-write-maintain-read-memp
 $\text{read-memp}(\text{addr}, \text{byte-write}(\text{value}, x, \text{mem}), n) = \text{read-memp}(\text{addr}, \text{mem}, n)$

THEOREM: byte-write-maintain-write-memp
 $\text{write-memp}(\text{addr}, \text{byte-write}(\text{value}, x, \text{mem}), n) = \text{write-memp}(\text{addr}, \text{mem}, n)$

THEOREM: write-mem-maintain-pc-read-memp
 $\text{pc-read-memp}(\text{addr}, \text{write-mem}(\text{value}, x, \text{mem}, m), n) = \text{pc-read-memp}(\text{addr}, \text{mem}, n)$

THEOREM: write-mem-maintain-read-memp
 $\text{read-memp}(\text{addr}, \text{write-mem}(\text{value}, x, \text{mem}, m), n) = \text{read-memp}(\text{addr}, \text{mem}, n)$

THEOREM: write-mem-maintain-write-memp
 $\text{write-memp}(\text{addr}, \text{write-mem}(\text{value}, x, \text{mem}, m), n) = \text{write-memp}(\text{addr}, \text{mem}, n)$

; if it is pc-readable, then it must be readable.

THEOREM: pc-read-memp->read-memp
 $\text{pc-read-memp}(x, \text{mem}, n) \rightarrow \text{read-memp}(x, \text{mem}, n)$

; if it is writeable, then it must be readable.

THEOREM: write-memp->read-memp
 $\text{write-memp}(x, \text{mem}, n) \rightarrow \text{read-memp}(x, \text{mem}, n)$

; program segments are never changed as the memory are changed.

THEOREM: pc-byte-read-write-mem
 $(\text{write-memp}(x, \text{mem}, n) \wedge \text{pc-byte-readp}(y, \text{mem}))$
 $\rightarrow (\text{pc-byte-read}(y, \text{write-mem}(\text{value}, x, \text{mem}, n)) = \text{pc-byte-read}(y, \text{mem}))$

THEOREM: pc-read-mem-byte-write
 $(\text{byte-writep}(x, \text{mem}) \wedge \text{pc-read-memp}(y, \text{mem}, n))$
 $\rightarrow (\text{pc-read-mem}(y, \text{byte-write}(\text{value}, x, \text{mem}), n) = \text{pc-read-mem}(y, \text{mem}, n))$

THEOREM: pc-read-mem-write-mem
 $(\text{write-memp}(x, \text{mem}, m) \wedge \text{pc-read-memp}(y, \text{mem}, n))$
 $\rightarrow (\text{pc-read-mem}(y, \text{write-mem}(\text{value}, x, \text{mem}, m), n) = \text{pc-read-mem}(y, \text{mem}, n))$

; a byte-read vs a multi-write.
; stop proving.

DEFINITION:

byte-read-write-mem-end (x , $value$, y , mem , k)
= byte-read (x , write-mem ($value$, y , mem , k))

THEOREM: byte-read-write-mem

byte-read (x , write-mem ($value$, y , mem , k))
= if disjoint (x , 1, y , k) then byte-read (x , mem)
else byte-read-write-mem-end (x , $value$, y , mem , k) endif

; a multi-read vs a byte-write.
; stop proving.

DEFINITION:

read-mem-byte-write-end (x , n , $value$, y , mem)
= read-mem (x , byte-write ($value$, y , mem), n)

THEOREM: read-mem-byte-write

read-mem (x , byte-write ($value$, y , mem), n)
= if disjoint0 (x , n , y) then read-mem (x , mem , n)
else read-mem-byte-write-end (x , n , $value$, y , mem) endif

; a multi-read vs a multi-write.
; stop proving.

DEFINITION:

read-write-mem-end (x , $value$, y , mem , m , n)
= read-mem (x , write-mem ($value$, y , mem , m), n)

THEOREM: read-write-mem1

read-mem (x , write-mem ($value$, y , mem , k), n)
= if disjoint (x , n , y , k) then read-mem (x , mem , n)
else read-write-mem-end (x , $value$, y , mem , k , n) endif

THEOREM: head-sub1-lessp

((head (x , n) - 1) < x) = ($x \neq 0$)

THEOREM: byte-read-write-mem-lemma

(nat-range (y , 32) $\wedge$ ($n \leq y$))
→ (byte-read (add (32, x , y), write-mem ($value$, x , mem , n))
= byte-read (add (32, x , y), mem))

THEOREM: read-write-mem2

uint-range (n , 32)
→ (read-mem (x , write-mem ($value$, x , mem , n), n) = head ($value$, 8 * n))

EVENT: Disable read-write-mem-end.

EVENT: Disable read-mem-byte-write-end.

EVENT: Disable byte-read-write-mem-end.

; write to the same location twice, only the second counts.

THEOREM: write-mem-byte-write

write-mem($v2, y, \text{byte-write}(v1, x, \text{mem}), n$)
= if disjoint0(y, n, x) then byte-write($v1, x, \text{write-mem}(v2, y, \text{mem}, n)$)
else write-mem($v2, y, \text{mem}, n$) endif

DEFINITION:

write-write-induct($v1, v2, n$)
= if $n \simeq 0$ then t
else write-write-induct($\text{tail}(v1, B), \text{tail}(v2, B), n - 1$) endif

THEOREM: write-write-mem

write-mem($v2, x, \text{write-mem}(v1, x, \text{mem}, n), n$) = write-mem($v2, x, \text{mem}, n$)

; LEMMAS ABOUT BITP

THEOREM: bitp-fix-bit

bitp(x) $\rightarrow$ (fix-bit(x) = x)

THEOREM: fix-bit-bitp

bitp(fix-bit(b))

THEOREM: bitn-bitp

bitp(bitn(x, n))

THEOREM: add-c-bitp

bitp(add-c(n, x, y))

THEOREM: add-v-bitp

bitp(add-v(n, x, y))

THEOREM: add-z-bitp

bitp(add-z(n, x, y))

THEOREM: add-n-bitp

bitp(add-n(n, x, y))

THEOREM: addx-c-bitp
bitp (addx-c (n, x, s, d))

THEOREM: addx-v-bitp
bitp (addx-v (n, x, s, d))

THEOREM: addx-z-bitp
bitp (addx-z (n, z, x, s, d))

THEOREM: addx-n-bitp
bitp (addx-n (n, x, s, d))

THEOREM: sub-c-bitp
bitp (sub-c (n, x, y))

THEOREM: sub-v-bitp
bitp (sub-v (n, x, y))

THEOREM: sub-z-bitp
bitp (sub-z (n, x, y))

THEOREM: sub-n-bitp
bitp (sub-n (n, x, y))

THEOREM: subx-c-bitp
bitp (subx-c (n, x, s, d))

THEOREM: subx-v-bitp
bitp (subx-v (n, x, s, d))

THEOREM: subx-z-bitp
bitp (subx-z (n, z, x, s, d))

THEOREM: subx-n-bitp
bitp (subx-n (n, x, s, d))

THEOREM: and-z-bitp
bitp (and-z (n, s, d))

THEOREM: and-n-bitp
bitp (and-n (n, s, d))

THEOREM: mulu-v-bitp
bitp (mulu-v (n, s, d, i))

THEOREM: mulu-z-bitp
bitp (mulu-z (n, s, d, i))

THEOREM: mulu-n-bitp
bitp (mulu-n (n, s, d, i))

THEOREM: muls-v-bitp
bitp (muls-v (n, s, d, i))

THEOREM: muls-z-bitp
bitp (muls-z (n, s, d, i))

THEOREM: muls-n-bitp
bitp (muls-n (n, s, d, i))

THEOREM: or-z-bitp
bitp (or-z (n, s, d))

THEOREM: or-n-bitp
bitp (or-n (n, s, d))

THEOREM: divs-z-bitp
bitp (divs-z (n, s, i, d, j))

THEOREM: divs-n-bitp
bitp (divs-n (n, s, i, d, j))

THEOREM: divu-z-bitp
bitp (divu-z (n, s, d))

THEOREM: divu-n-bitp
bitp (divu-n (n, s, d))

THEOREM: rol-c-bitp
bitp (rol-c (len, x, cnt))

THEOREM: rol-z-bitp
bitp (rol-z (len, x, cnt))

THEOREM: rol-n-bitp
bitp (rol-n (len, x, cnt))

THEOREM: ror-c-bitp
bitp (ror-c (len, x, cnt))

THEOREM: ror-z-bitp
bitp (ror-z (len, x, cnt))

THEOREM: ror-n-bitp
bitp (ror-n (len, x, cnt))

THEOREM: lsl-c-bitp
 $\text{bitp}(\text{lsl-c}(len, x, cnt))$

THEOREM: lsl-z-bitp
 $\text{bitp}(\text{lsl-z}(len, x, cnt))$

THEOREM: lsl-n-bitp
 $\text{bitp}(\text{lsl-n}(len, x, cnt))$

THEOREM: lsr-c-bitp
 $\text{bitp}(\text{lsr-c}(len, x, cnt))$

THEOREM: lsr-z-bitp
 $\text{bitp}(\text{lsr-z}(len, x, cnt))$

THEOREM: lsr-n-bitp
 $\text{bitp}(\text{lsr-n}(len, x, cnt))$

THEOREM: asl-c-bitp
 $\text{bitp}(\text{asl-c}(len, x, cnt))$

THEOREM: asl-v-bitp
 $\text{bitp}(\text{asl-v}(len, x, cnt))$

THEOREM: asl-z-bitp
 $\text{bitp}(\text{asl-z}(len, x, cnt))$

THEOREM: asl-n-bitp
 $\text{bitp}(\text{asl-n}(len, x, cnt))$

THEOREM: asr-c-bitp
 $\text{bitp}(\text{asr-c}(len, x, cnt))$

THEOREM: asr-z-bitp
 $\text{bitp}(\text{asr-z}(len, x, cnt))$

THEOREM: asr-n-bitp
 $\text{bitp}(\text{asr-n}(len, x, cnt))$

THEOREM: roxl-c-bitp
 $\text{bitp}(\text{roxl-c}(len, opd, cnt, x))$

THEOREM: roxl-z-bitp
 $\text{bitp}(\text{roxl-z}(len, opd, cnt, x))$

THEOREM: roxl-n-bitp
 $\text{bitp}(\text{roxl-n}(len, opd, cnt, x))$

THEOREM: roxr-c-bitp
 $\text{bitp}(\text{roxr-c}(len, opd, cnt, x))$

THEOREM: roxr-z-bitp
 $\text{bitp}(\text{roxr-z}(len, opd, cnt, x))$

THEOREM: roxr-n-bitp
 $\text{bitp}(\text{roxr-n}(len, opd, cnt, x))$

THEOREM: move-z-bitp
 $\text{bitp}(\text{move-z}(oplen, sopd))$

THEOREM: move-n-bitp
 $\text{bitp}(\text{move-n}(oplen, sopd))$

THEOREM: ext-z-bitp
 $\text{bitp}(\text{ext-z}(oplen, opd, size))$

THEOREM: ext-n-bitp
 $\text{bitp}(\text{ext-n}(oplen, opd, size))$

THEOREM: swap-z-bitp
 $\text{bitp}(\text{swap-z}(opd))$

THEOREM: swap-n-bitp
 $\text{bitp}(\text{swap-n}(opd))$

THEOREM: not-z-bitp
 $\text{bitp}(\text{not-z}(oplen, opd))$

THEOREM: not-n-bitp
 $\text{bitp}(\text{not-n}(oplen, opd))$

THEOREM: eor-z-bitp
 $\text{bitp}(\text{eor-z}(n, s, d))$

THEOREM: eor-n-bitp
 $\text{bitp}(\text{eor-n}(n, s, d))$

; NEG
 ; after replacing sub by add and neg. We can completely get rid of sub.

THEOREM: neg-head
 $\text{neg}(n, \text{head}(x, n)) = \text{neg}(n, x)$

THEOREM: neg-neg
 $\text{neg}(n, \text{neg}(n, x)) = \text{head}(x, n)$

THEOREM: neg-cancel
 $\text{add}(n, x, \text{neg}(n, x)) = 0$

THEOREM: neg-add
 $\text{neg}(n, \text{add}(n, x, y)) = \text{add}(n, \text{neg}(n, x), \text{neg}(n, y))$

THEOREM: sub-neg
 $\text{sub}(n, y, x) = \text{add}(n, x, \text{neg}(n, y))$

EVENT: Disable neg.

; $x - y = 0 \iff x = y$.

THEOREM: add-neg-0
 $(\text{nat-range}(x, n) \wedge \text{nat-range}(y, n))$
 $\rightarrow ((\text{add}(n, y, \text{neg}(n, x)) = 0) = (\text{fix}(x) = \text{fix}(y)))$

; $y + (x - y) = x$.

THEOREM: add-neg-cancel
 $(\text{add}(n, y, \text{add}(n, x, \text{neg}(n, y))) = \text{head}(x, n))$
 $\wedge (\text{add}(n, y, \text{add}(n, \text{neg}(n, y), x)) = \text{head}(x, n))$

THEOREM: sub-leq-1
 $(\text{neg}(n, x) \leq y)$
 $\rightarrow (((y - \text{neg}(n, x)) \not\leq \text{add}(n, y, x))$
 $\wedge ((y - \text{neg}(n, x)) \not\leq \text{add}(n, x, y)))$

THEOREM: sub-leq-2
 $(x \leq y)$
 $\rightarrow (((y - x) \not\leq \text{add}(n, y, \text{neg}(n, x)))$
 $\wedge ((y - x) \not\leq \text{add}(n, \text{neg}(n, x), y)))$

DEFINITION:
 $\text{add-fringe}(n, x)$
 $=$ **if** $\text{listp}(x) \wedge (\text{car}(x) = \text{'add}) \wedge (\text{cadr}(x) = n)$
then $\text{append}(\text{add-fringe}(n, \text{caddr}(x)), \text{add-fringe}(n, \text{cadddr}(x)))$
else $\text{cons}(x, \text{nil})$ **endif**

DEFINITION:
 $\text{add-tree}(n, l)$
 $=$ **if** $l \simeq \text{nil}$ **then** $'0$
elseif $\text{cdr}(l) \simeq \text{nil}$ **then** $\text{list}(\text{'head}, \text{car}(l), n)$
elseif $\text{caddr}(l) \simeq \text{nil}$ **then** $\text{list}(\text{'add}, n, \text{car}(l), \text{cadr}(l))$
else $\text{list}(\text{'add}, n, \text{car}(l), \text{add-tree}(n, \text{cdr}(l)))$ **endif**

THEOREM: numberp-eval\$-add
 $(\text{car}(x) = \text{'add}) \rightarrow (\text{eval\$}(\mathbf{t}, x, a) \in \mathbf{N})$

THEOREM: numberp-eval\$-add-tree
 $\text{eval\$}(\mathbf{t}, \text{add-tree}(n, l), a) \in \mathbf{N}$

THEOREM: add-equal-cancel-1
 $(\text{add}(n, a, b) = \text{add}(n, c, \text{add}(n, a, d))) = (\text{head}(b, n) = \text{add}(n, c, d))$

THEOREM: eval\$-add-member
 $(e \in x)$
 $\rightarrow (\text{eval\$}(\mathbf{t}, \text{add-tree}(n, x), a)$
 $\quad = \text{add}(\text{eval\$}(\mathbf{t}, n, a),$
 $\quad \quad \text{eval\$}(\mathbf{t}, e, a),$
 $\quad \quad \text{eval\$}(\mathbf{t}, \text{add-tree}(n, \text{delete}(e, x)), a)))$

THEOREM: add-tree-nat-range
 $\text{nat-range}(\text{eval\$}(\mathbf{t}, \text{add-tree}(n, x), a), \text{eval\$}(\mathbf{t}, n, a))$

THEOREM: add-tree-append
 $\text{eval\$}(\mathbf{t}, \text{add-tree}(n, \text{append}(x, y)), a)$
 $= \text{add}(\text{eval\$}(\mathbf{t}, n, a), \text{eval\$}(\mathbf{t}, \text{add-tree}(n, x), a), \text{eval\$}(\mathbf{t}, \text{add-tree}(n, y), a))$

THEOREM: add-tree-add-fringe
 $\text{eval\$}(\mathbf{t}, \text{add-tree}(n, \text{add-fringe}(n, x)), a) = \text{head}(\text{eval\$}(\mathbf{t}, x, a), \text{eval\$}(\mathbf{t}, n, a))$

DEFINITION:
 $\text{add-tree-delete-cond}(n, x, y)$
 $= \text{list}(\text{'and},$
 $\quad \text{list}(\text{'numberp}, x),$
 $\quad \text{list}(\text{'and},$
 $\quad \quad \text{list}(\text{'nat-range}, x, n),$
 $\quad \quad \text{list}(\text{'equal}, \text{add-tree}(n, \text{delete}(x, y)), \text{'0})))$

THEOREM: add-tree-delete-equal
 $(x \in y)$
 $\rightarrow (\text{eval\$}(\mathbf{t}, \text{add-tree-delete-cond}(n, x, y), a)$
 $\quad = (\text{eval\$}(\mathbf{t}, \text{add-tree}(n, y), a) = \text{eval\$}(\mathbf{t}, x, a)))$

THEOREM: add-tree-bagdiff
 $(\text{subbagp}(x, y) \wedge \text{subbagp}(x, z))$
 $\rightarrow ((\text{eval\$}(\mathbf{t}, \text{add-tree}(n, \text{bagdiff}(y, x)), a)$
 $\quad = \text{eval\$}(\mathbf{t}, \text{add-tree}(n, \text{bagdiff}(z, x)), a))$
 $\quad = (\text{eval\$}(\mathbf{t}, \text{add-tree}(n, y), a) = \text{eval\$}(\mathbf{t}, \text{add-tree}(n, z), a)))$

DEFINITION:

```

cancel-equal-add (x)
=  if listp (x) ∧ (car (x) = 'equal)
    then if listp (cadr (x))
        ∧ (caadr (x) = 'add)
        ∧ listp (caddr (x))
        ∧ (caaddr (x) = 'add)
        then if cadadr (x) = cadaddr (x)
            then list ('equal,
                add-tree (cadadr (x),
                    bagdiff (add-fringe (cadadr (x), cadr (x)),
                        bagint (add-fringe (cadadr (x),
                            cadr (x)),
                                add-fringe (cadadr (x),
                                    caddr (x))))),
                    add-tree (cadadr (x),
                        bagdiff (add-fringe (cadadr (x), caddr (x)),
                            bagint (add-fringe (cadadr (x),
                                cadr (x)),
                                    add-fringe (cadadr (x),
                                        caddr (x))))))
                else x endif
        elseif listp (cadr (x))
            ∧ (caadr (x) = 'add)
            ∧ (caddr (x) ∈ add-fringe (cadadr (x), cadr (x)))
            then add-tree-delete-cond (cadadr (x),
                cadr (x),
                add-fringe (cadadr (x), cadr (x)))
        elseif listp (caddr (x))
            ∧ (caaddr (x) = 'add)
            ∧ (cadr (x) ∈ add-fringe (cadaddr (x), caddr (x)))
            then add-tree-delete-cond (cadaddr (x),
                cadr (x),
                add-fringe (cadaddr (x), caddr (x)))
        else x endif
    else x endif

```

THEOREM: correctness-of-cancel-equal-add

$$\text{eval\$}(\mathbf{t}, x, a) = \text{eval\$}(\mathbf{t}, \text{cancel-equal-add}(x), a)$$

THEOREM: add-tree-delete

$$\begin{aligned}
 & (x \in y) \\
 \rightarrow & (\text{eval\$}(\mathbf{t}, \text{add-tree}(n, \text{delete}(x, y)), a) \\
 & = \text{sub}(\text{eval\$}(\mathbf{t}, n, a), \text{eval\$}(\mathbf{t}, x, a), \text{eval\$}(\mathbf{t}, \text{add-tree}(n, y), a)))
 \end{aligned}$$

DEFINITION:

cancel-add-neg (x)

```
=  if listp( $x$ )
    ∧ (car( $x$ ) = 'add)
    ∧ listp(cdr( $x$ ))
    ∧ listp(caddr( $x$ ))
    then if listp(caddr( $x$ ))
        ∧ listp(cdaddr( $x$ ))
        ∧ listp(cddaddr( $x$ ))
        ∧ (caaddr( $x$ ) = 'neg)
        ∧ (cadaddr( $x$ ) = cadr( $x$ ))
        ∧ (caddaddr( $x$ ) ∈ add-fringe(cadr( $x$ ), caddr( $x$ )))
        then add-tree(cadr( $x$ ),
            delete(caddaddr( $x$ ), add-fringe(cadr( $x$ ), caddr( $x$ ))))
        elseif list('neg, cadr( $x$ ), caddr( $x$ ))
            ∈ add-fringe(cadr( $x$ ), caddr( $x$ ))
            then add-tree(cadr( $x$ ),
                delete(list('neg, cadr( $x$ ), caddr( $x$ )),
                    add-fringe(cadr( $x$ ), caddr( $x$ ))))
    else  $x$  endif
else  $x$  endif
```

THEOREM: correctness-of-cancel-add-neg

eval\$(t, x, a) = eval\$(t , cancel-add-neg(x), a)

EVENT: For efficiency, compile those definitions not yet compiled.

```
;          CONDITION CODES
; this section considers the various condition codes in Bcc and Scc
; instructions. The condition codes are specified as follows:
; CC carry clear      ~C          LS low or same      C+Z
; CS carry set        C           LT less than        N*~V+~N*V
; EQ equal            Z           MI minus            N
; F never true        0           NE not equal        ~Z
; GE greater or equal N*V+~N*~V   PL plus            ~N
; GT greater than     N*V*~Z+~N*~V*~Z T always true  1
; HI high             ~C*~Z       VC overflow clear  ~V
; LE less or equal    Z+N*~V+~N*V VS overflow set    V

;          BHI/BLS
; two bridge lemmas.
; the relation between add-c and addx-c.
```

THEOREM: add-addx-c

add-c(n, x, y) = addx-c($n, 0, x, y$)

; the relation between sub-c and subx-c.

THEOREM: sub-subx-c

$$\text{sub-c}(n, x, y) = \text{subx-c}(n, 0, x, y)$$

; two lemmas for addx-c and subx-c.

THEOREM: addx-c-la

$$\begin{aligned} & (\text{nat-range}(x, n) \wedge \text{nat-range}(y, n) \wedge \text{bitp}(c) \wedge (n \neq 0)) \\ \rightarrow & \quad (\text{addx-c}(n, c, x, y) \\ & = \quad \text{if } (c + x + y) < \text{exp}(2, n) \text{ then } 0 \\ & \quad \text{else } 1 \text{ endif}) \end{aligned}$$

THEOREM: subx-c-la

$$\begin{aligned} & (\text{nat-range}(x, n) \wedge \text{nat-range}(y, n) \wedge \text{bitp}(c) \wedge (n \neq 0)) \\ \rightarrow & \quad (\text{subx-c}(n, c, x, y) \\ & = \quad \text{if } y < (c + x) \text{ then } 1 \\ & \quad \text{else } 0 \text{ endif}) \end{aligned}$$

EVENT: Disable addx-c.

EVENT: Disable add-c.

EVENT: Disable subx-c.

EVENT: Disable sub-c.

EVENT: Disable mbit-means-lessp.

THEOREM: sub-z-la

$$\begin{aligned} & (\text{nat-range}(x, n) \wedge \text{nat-range}(y, n)) \\ \rightarrow & \quad (\text{sub-z}(n, x, y) \\ & = \quad \text{if } \text{fix}(x) = \text{fix}(y) \text{ then } 1 \\ & \quad \text{else } 0 \text{ endif}) \end{aligned}$$

DEFINITION: $\text{between-ileq}(x, y, z) = (\text{ileq}(x, y) \wedge \text{ileq}(y, z))$

THEOREM: sub-bhi-int

$$\begin{aligned} & (\text{nat-range}(x, n) \\ & \wedge \quad \text{nat-range}(y, n) \\ & \wedge \quad (n \neq 0) \\ & \wedge \quad (\neg \text{negativep}(\text{nat-to-int}(x, n)))) \\ \rightarrow & \quad (\text{bhi}(\text{sub-c}(n, x, y), \text{sub-z}(n, x, y)) \\ & = \quad \text{if } \text{between-ileq}(0, \text{nat-to-int}(y, n), \text{nat-to-int}(x, n)) \text{ then } 0 \\ & \quad \text{else } 1 \text{ endif}) \end{aligned}$$

THEOREM: sub-bhi-0

$$\begin{aligned}
& (\text{nat-range}(x, n) \\
& \wedge \text{nat-range}(y, n) \\
& \wedge (\text{nat-to-uint}(x) \not< \text{nat-to-uint}(y)) \\
& \wedge (n \neq 0)) \\
& \rightarrow (\text{bhi}(\text{sub-c}(n, x, y), \text{sub-z}(n, x, y)) = 0)
\end{aligned}$$

THEOREM: sub-bhi-1

$$\begin{aligned}
& (\text{nat-range}(x, n) \\
& \wedge \text{nat-range}(y, n) \\
& \wedge (\text{nat-to-uint}(x) < \text{nat-to-uint}(y)) \\
& \wedge (n \neq 0)) \\
& \rightarrow (\text{bhi}(\text{sub-c}(n, x, y), \text{sub-z}(n, x, y)) = 1)
\end{aligned}$$

THEOREM: sub-bls

$$\begin{aligned}
& (\text{nat-range}(x, n) \wedge \text{nat-range}(y, n) \wedge (n \neq 0)) \\
& \rightarrow (\text{bls}(\text{sub-c}(n, x, y), \text{sub-z}(n, x, y)) \\
& \quad = \text{if } \text{nat-to-uint}(x) < \text{nat-to-uint}(y) \text{ then } 0 \\
& \quad \text{else } 1 \text{ endif})
\end{aligned}$$

; the trivial relation between bls and bhi.

THEOREM: bls-bhi

$$\text{bls}(c, z) = \text{b-not}(\text{bhi}(c, z))$$

EVENT: Disable bhi.

EVENT: Disable bls.

; BEQ/BNE
; the z-flag of move.
; the unsigned integer interpretation.

THEOREM: move-beq-uint

$$\begin{aligned}
& \text{nat-range}(x, n) \\
& \rightarrow (\text{beq}(\text{move-z}(n, x)) \\
& \quad = \text{if } \text{nat-to-uint}(x) = 0 \text{ then } 1 \\
& \quad \text{else } 0 \text{ endif})
\end{aligned}$$

; the signed integer interpretation.

THEOREM: move-beq-int-0

$$(\text{nat-range}(x, n) \wedge (\text{nat-to-int}(x, n) \neq 0)) \rightarrow (\text{beq}(\text{move-z}(n, x)) = 0)$$

THEOREM: move-beq-int-1

$(\text{nat-range}(x, n) \wedge (\text{nat-to-int}(x, n) = 0)) \rightarrow (\text{beq}(\text{move-z}(n, x)) = 1)$

; zero testing on a sign-extended bit vector is equivalent to
; testing on the original value.

THEOREM: move-beq-ext

$(\text{nat-range}(x, n) \wedge (n \leq \text{size}) \wedge (n \neq 0))$
 $\rightarrow (\text{move-z}(\text{size}, \text{ext}(n, x, \text{size})) = \text{move-z}(n, x))$

; the z-flag of sub.
; the unsigned integer interpretation.

THEOREM: sub-beq-uint

$(\text{nat-range}(x, n) \wedge \text{nat-range}(y, n))$
 $\rightarrow (\text{beq}(\text{sub-z}(n, x, y))$
 $\quad = \text{if } \text{nat-to-uint}(x) = \text{nat-to-uint}(y) \text{ then } 1$
 $\quad \text{else } 0 \text{ endif})$

; the signed integer interpretation.

THEOREM: sub-beq-int-0

$(\text{nat-range}(x, n)$
 $\quad \wedge \text{nat-range}(y, n)$
 $\quad \wedge (\text{nat-to-int}(x, n) \neq \text{nat-to-int}(y, n)))$
 $\rightarrow (\text{beq}(\text{sub-z}(n, x, y)) = 0)$

THEOREM: sub-beq-int-1

$(\text{nat-range}(x, n)$
 $\quad \wedge \text{nat-range}(y, n)$
 $\quad \wedge (\text{nat-to-int}(x, n) = \text{nat-to-int}(y, n)))$
 $\rightarrow (\text{beq}(\text{sub-z}(n, x, y)) = 1)$

; equality testing on two sign-extended bit vectors is equivalent to
; testing on their original values.

THEOREM: sub-beq-ext

$(\text{nat-range}(x, n) \wedge \text{nat-range}(y, n) \wedge (n \leq \text{size}) \wedge (n \neq 0))$
 $\rightarrow (\text{sub-z}(\text{size}, \text{ext}(n, x, \text{size}), \text{ext}(n, y, \text{size})) = \text{sub-z}(n, x, y))$

; the z flag of logor.

THEOREM: logor-beq-uint

$\text{beq}(\text{or-z}(n, x, y))$
 $= \text{if } (\text{nat-to-uint}(x) = 0) \wedge (\text{nat-to-uint}(y) = 0) \text{ then } 1$
 $\quad \text{else } 0 \text{ endif}$

THEOREM: logor-beq-int-0

$$\begin{aligned} & (\text{nat-range}(x, n) \\ & \wedge \text{nat-range}(y, n) \\ & \wedge ((\text{nat-to-int}(x, n) \neq 0) \vee (\text{nat-to-int}(y, n) \neq 0))) \\ & \rightarrow (\text{beq}(\text{or-z}(n, x, y)) = 0) \end{aligned}$$

THEOREM: logor-beq-int-1

$$\begin{aligned} & (\text{nat-range}(x, n) \\ & \wedge \text{nat-range}(y, n) \\ & \wedge (\text{nat-to-int}(x, n) = 0) \\ & \wedge (\text{nat-to-int}(y, n) = 0)) \\ & \rightarrow (\text{beq}(\text{or-z}(n, x, y)) = 1) \end{aligned}$$

; the z flag of logeor.

THEOREM: logeor-beq-uint

$$\begin{aligned} & \text{beq}(\text{eor-z}(n, x, y)) \\ & = \text{if } \text{nat-to-uint}(x) = \text{nat-to-uint}(y) \text{ then } 1 \\ & \quad \text{else } 0 \text{ endif} \end{aligned}$$

THEOREM: logeor-beq-int-0

$$\begin{aligned} & (\text{nat-range}(x, n) \\ & \wedge \text{nat-range}(y, n) \\ & \wedge (\text{nat-to-int}(x, n) \neq \text{nat-to-int}(y, n))) \\ & \rightarrow (\text{beq}(\text{eor-z}(n, x, y)) = 0) \end{aligned}$$

THEOREM: logeor-beq-int-1

$$\begin{aligned} & (\text{nat-range}(x, n) \\ & \wedge \text{nat-range}(y, n) \\ & \wedge (\text{nat-to-int}(x, n) = \text{nat-to-int}(y, n))) \\ & \rightarrow (\text{beq}(\text{eor-z}(n, x, y)) = 1) \end{aligned}$$

; the z flag of logand.

THEOREM: logand-beq-uint

$$\begin{aligned} & (\exp(2, \log(2, 1 + x)) = (1 + x)) \\ & \rightarrow (\text{beq}(\text{and-z}(n, x, y)) \\ & \quad = \text{if } (\text{nat-to-uint}(y) \bmod (1 + x)) = 0 \text{ then } 1 \\ & \quad \text{else } 0 \text{ endif}) \end{aligned}$$

THEOREM: logand-or-beq-uint

$$\begin{aligned} & (\exp(2, \log(2, 1 + x)) = (1 + x)) \\ & \rightarrow (\text{beq}(\text{and-z}(n, x, \text{logor}(y, z))) \\ & \quad = \text{if } ((\text{nat-to-uint}(y) \bmod (1 + x)) = 0) \\ & \quad \wedge ((\text{nat-to-uint}(z) \bmod (1 + x)) = 0) \text{ then } 1 \\ & \quad \text{else } 0 \text{ endif}) \end{aligned}$$

THEOREM: logand-eor-beq-uint

$$\begin{aligned} & (\exp(2, \log(2, 1 + x)) = (1 + x)) \\ \rightarrow & (\text{beq}(\text{and-z}(n, x, \text{logeor}(y, z))) \\ & = \text{if}(\text{nat-to-uint}(y) \bmod (1 + x)) \\ & = (\text{nat-to-uint}(z) \bmod (1 + x)) \text{ then } 1 \\ & \text{else } 0 \text{ endif}) \end{aligned}$$

; the z-flag of mulu. There are three cases to handle.

THEOREM: mulu_1632-beq

$$\begin{aligned} & (\text{nat-range}(x, 16) \wedge \text{nat-range}(y, 16)) \\ \rightarrow & (\text{beq}(\text{mulu-z}(32, x, y, 16)) \\ & = \text{if}(\text{nat-to-uint}(x) = 0) \vee (\text{nat-to-uint}(y) = 0) \text{ then } 1 \\ & \text{else } 0 \text{ endif}) \end{aligned}$$

THEOREM: mulu_3264-beq

$$\begin{aligned} & (\text{nat-range}(x, 32) \wedge \text{nat-range}(y, 32)) \\ \rightarrow & (\text{beq}(\text{mulu-z}(64, x, y, 32)) \\ & = \text{if}(\text{nat-to-uint}(x) = 0) \vee (\text{nat-to-uint}(y) = 0) \text{ then } 1 \\ & \text{else } 0 \text{ endif}) \end{aligned}$$

THEOREM: mulu_3232-beq

$$\begin{aligned} & \text{uint-range}(\text{nat-to-uint}(x) * \text{nat-to-uint}(y), 32) \\ \rightarrow & (\text{beq}(\text{mulu-z}(32, x, y, 32)) \\ & = \text{if}(\text{nat-to-uint}(x) = 0) \vee (\text{nat-to-uint}(y) = 0) \text{ then } 1 \\ & \text{else } 0 \text{ endif}) \end{aligned}$$

; the z-flag of muls. There are three cases to handle.

THEOREM: muls_1632-beq

$$\begin{aligned} & (\text{nat-range}(x, 16) \wedge \text{nat-range}(y, 16)) \\ \rightarrow & (\text{beq}(\text{muls-z}(32, x, y, 16)) \\ & = \text{if}(\text{nat-to-int}(x, 16) = 0) \vee (\text{nat-to-int}(y, 16) = 0) \\ & \text{then } 1 \\ & \text{else } 0 \text{ endif}) \end{aligned}$$

THEOREM: muls_3264-beq

$$\begin{aligned} & (\text{nat-range}(x, 32) \wedge \text{nat-range}(y, 32)) \\ \rightarrow & (\text{beq}(\text{muls-z}(64, x, y, 32)) \\ & = \text{if}(\text{nat-to-int}(x, 32) = 0) \vee (\text{nat-to-int}(y, 32) = 0) \\ & \text{then } 1 \\ & \text{else } 0 \text{ endif}) \end{aligned}$$

THEOREM: muls_3232-beq

$$(\text{int-range}(\text{itimes}(\text{nat-to-int}(x, 32), \text{nat-to-int}(y, 32)), 32))$$

```

 $\wedge$  nat-range $p(x, 32)$ 
 $\wedge$  nat-range $p(y, 32)$ 
 $\rightarrow$  (beq(muls-z(32, x, y, 32))
      = if (nat-to-int(x, 32) = 0)  $\vee$  (nat-to-int(y, 32) = 0)
          then 1
          else 0 endif)

```

; the z-flag of divu.

THEOREM: divu-beq
 (nat-range $p(y, n) \wedge$ (nat-to-uint(x) $\neq$ 0))
 $\rightarrow$ (beq(divu-z(n, x, y))
 = if nat-to-uint(y) < nat-to-uint(x) then 1
 else 0 endif)

; the z-flag of divs.

THEOREM: iquotient=0
 (integerp(y) $\wedge$ (y $\neq$ 0))
 $\rightarrow$ ((iquotient(x, y) = 0) = (abs(x) < abs(y)))

THEOREM: divs-beq
 (int-range p (iquotient(nat-to-int(y, j), nat-to-int(x, i)), n)
 $\wedge$ nat-range $p(x, i)$
 $\wedge$ (nat-to-int(x, i) $\neq$ 0)
 $\wedge$ (n $\leq$ j))
 $\rightarrow$ (beq(divs-z(n, x, i, y, j))
 = if abs(nat-to-int(y, j)) < abs(nat-to-int(x, i)) then 1
 else 0 endif)

; the z-flag of lsl.

THEOREM: lsl-beq
 uint-range p (nat-to-uint(x), n - cnt)
 $\rightarrow$ (beq(lsl-z(n, x, cnt))
 = if nat-to-uint(x) = 0 then 1
 else 0 endif)

; the z-flag of lsr.

THEOREM: lsr-beq
 beq(lsr-z(n, x, cnt))
 = if nat-to-uint(x) < exp(2, cnt) then 1
 else 0 endif

THEOREM: z-flag-la
 (x $\in$ $\mathbf{N}$) $\rightarrow$ ((nat-to-int(x, n) = 0) = (x = 0))

; the z-flag of asl.

THEOREM: asl-beq
(nat-range(x, n) $\wedge$ int-range($\text{nat-to-int}(x, n), n - cnt$))
 $\rightarrow$ (beq($\text{asl-z}(n, x, cnt)$)
= if $\text{nat-to-int}(x, n) = 0$ then 1
else 0 endif)

; the z-flag of asr.

THEOREM: asr-beq
nat-range(x, n)
 $\rightarrow$ (beq($\text{asr-z}(n, x, cnt)$)
= if negativep($\text{nat-to-int}(x, n)$) then 0
elseif $\text{nat-to-int}(x, n) < \text{exp}(2, cnt)$ then 1
else 0 endif)

; the z-flag of ext.

THEOREM: ext-beq-uint
(nat-range(x, n) $\wedge$ ($n \leq size$))
 $\rightarrow$ (beq($\text{ext-z}(n, x, size)$)
= if $\text{nat-to-uint}(x) = 0$ then 1
else 0 endif)

THEOREM: ext-beq-int-0
(nat-range(x, n) $\wedge$ ($n < size$) $\wedge$ ($\text{nat-to-int}(x, n) \neq 0$))
 $\rightarrow$ (beq($\text{ext-z}(n, x, size)$) = 0)

THEOREM: ext-beq-int-1
(nat-range(x, n) $\wedge$ ($n < size$) $\wedge$ ($\text{nat-to-int}(x, n) = 0$))
 $\rightarrow$ (beq($\text{ext-z}(n, x, size)$) = 1)

EVENT: Disable iquotient=0.

EVENT: Disable int-to-nat=0.

EVENT: Disable beq.

; BCS/BCC
; BCS/BCC means carry is set/cleared.
; the c-flag of sub.

THEOREM: sub-bcs&cc

$$\begin{aligned}
 & (\text{nat-range}(x, n) \wedge \text{nat-range}(y, n) \wedge (n \neq 0)) \\
 \rightarrow & (\text{bcs}(\text{sub-c}(n, x, y))) \\
 = & \text{if } \text{nat-to-uint}(y) < \text{nat-to-uint}(x) \text{ then } 1 \\
 & \text{else } 0 \text{ endif}
 \end{aligned}$$

; the c-flag of add.

THEOREM: add-bcs&cc

$$\begin{aligned}
 & (\text{nat-range}(x, n) \wedge \text{nat-range}(y, n) \wedge (n \neq 0)) \\
 \rightarrow & (\text{bcs}(\text{add-c}(n, x, y))) \\
 = & \text{if } (\text{nat-to-uint}(x) + \text{nat-to-uint}(y)) < \exp(2, n) \text{ then } 0 \\
 & \text{else } 1 \text{ endif}
 \end{aligned}$$

THEOREM: tail-mbit

$$\begin{aligned}
 & (x < \exp(2, n)) \\
 \rightarrow & (\text{tail}(x, n - 1)) \\
 = & \text{if } x < \exp(2, n - 1) \text{ then } 0 \\
 & \text{else } 1 \text{ endif}
 \end{aligned}$$

; the c-flag of lsl. But we only consider special cases: cnt = 0 or 1.

; I don't know what the meaning of c-flag is when cnt > 1.

THEOREM: lsl-c-0

$$\text{lsl-c}(n, x, 0) = 0$$

THEOREM: lsl-1-bcs&cc

$$\begin{aligned}
 & (\text{nat-range}(x, n) \wedge (1 < n)) \\
 \rightarrow & (\text{bcs}(\text{lsl-c}(n, x, 1))) \\
 = & \text{if } \text{uint-range}(\text{nat-to-uint}(x), n - 1) \text{ then } 0 \\
 & \text{else } 1 \text{ endif}
 \end{aligned}$$

; the c-flag of lsr. But we only consider special cases: cnt = 0 or 1.

; I don't know what the meaning of c-flag is when cnt > 1.

THEOREM: lsr-c-0

$$\text{lsr-c}(n, x, 0) = 0$$

THEOREM: lsr-1-bcs&cc

$$(1 < n) \rightarrow (\text{bcs}(\text{lsr-c}(n, x, 1)) = (\text{nat-to-uint}(x) \bmod 2))$$

EVENT: Disable tail-mbit.

EVENT: Disable bcs.

```

;                                BVS/BVC
; BVS/BVC means overflow is set/cleared.
; three bridge lemmas.

```

THEOREM: add-addx-v
 $\text{add-v}(n, x, y) = \text{addx-v}(n, 0, x, y)$

THEOREM: subx-addx-v
 $(\text{nat-range}(x, n) \wedge (n \neq 0))$
 $\rightarrow (\text{subx-v}(n, z, x, y) = \text{addx-v}(n, \text{b-not}(z), y, \text{lognot}(n, x)))$

THEOREM: sub-subx-v
 $\text{sub-v}(n, x, y) = \text{subx-v}(n, 0, x, y)$

```

; lemmas for addx-v and subx-v.

```

THEOREM: mbit-means-negativep
 $\text{nat-range}(x, n)$
 $\rightarrow (\text{bitn}(x, n - 1)$
 $= \text{if negativep}(\text{nat-to-int}(x, n)) \text{ then } 1$
 $\text{else } 0 \text{ endif})$

THEOREM: addx-v-la
 $(\text{nat-range}(x, n) \wedge \text{nat-range}(y, n) \wedge \text{bitp}(c) \wedge (n \neq 0))$
 $\rightarrow (\text{addx-v}(n, c, x, y)$
 $= \text{if int-range}(\text{iplus}(\text{nat-to-int}(x, n), \text{iplus}(\text{nat-to-int}(y, n), c)),$
 $n) \text{ then } 0$
 $\text{else } 1 \text{ endif})$

THEOREM: subx-v-la
 $(\text{nat-range}(x, n) \wedge \text{nat-range}(y, n) \wedge \text{bitp}(c) \wedge (n \neq 0))$
 $\rightarrow (\text{subx-v}(n, c, x, y)$
 $= \text{if int-range}(\text{idifference}(\text{nat-to-int}(y, n),$
 $\text{iplus}(\text{nat-to-int}(x, n), c)),$
 $n) \text{ then } 0$
 $\text{else } 1 \text{ endif})$

EVENT: Disable addx-v-crock1.

EVENT: Disable addx-v-crock2.

```

; the v-flag of sub.

```

THEOREM: sub-bvs&vc
 $(\text{nat-range}(x, n) \wedge \text{nat-range}(y, n) \wedge (n \neq 0))$
 $\rightarrow (\text{bvs}(\text{sub-v}(n, x, y)))$
 $= \text{if int-range}(\text{idifference}(\text{nat-to-int}(y, n), \text{nat-to-int}(x, n)), n)$
 $\quad \text{then } 0$
 $\quad \text{else } 1 \text{ endif}$

; the v-flag of add.

THEOREM: add-bvs&vc
 $(\text{nat-range}(x, n) \wedge \text{nat-range}(y, n) \wedge (n \neq 0))$
 $\rightarrow (\text{bvs}(\text{add-v}(n, x, y)))$
 $= \text{if int-range}(\text{iplus}(\text{nat-to-int}(x, n), \text{nat-to-int}(y, n)), n)$
 $\quad \text{then } 0$
 $\quad \text{else } 1 \text{ endif}$

; three cases for the v-flag of mulu.

THEOREM: mulu_1632-bvs
 $(\text{nat-range}(x, 16) \wedge \text{nat-range}(y, 16)) \rightarrow (\text{mulu-v}(32, x, y, 16) = 0)$

THEOREM: mulu_3264-bvs
 $(\text{nat-range}(x, 32) \wedge \text{nat-range}(y, 32)) \rightarrow (\text{mulu-v}(64, x, y, 32) = 0)$

THEOREM: mulu_3232-bvs
 $\text{bvs}(\text{mulu-v}(32, x, y, 32))$
 $= \text{if } (\text{nat-to-uint}(x) * \text{nat-to-uint}(y)) < \text{exp}(2, 32) \text{ then } 0$
 $\quad \text{else } 1 \text{ endif}$

; three cases for the v-flag of muls.

THEOREM: muls_1632-bvs
 $(\text{nat-range}(x, 16) \wedge \text{nat-range}(y, 16)) \rightarrow (\text{muls-v}(32, x, y, 16) = 0)$

THEOREM: muls_3264-bvs
 $(\text{nat-range}(x, 32) \wedge \text{nat-range}(y, 32)) \rightarrow (\text{muls-v}(64, x, y, 32) = 0)$

THEOREM: muls_3232-bvs
 $\text{bvs}(\text{muls-v}(32, x, y, 32))$
 $= \text{if int-range}(\text{itimes}(\text{nat-to-int}(x, 32), \text{nat-to-int}(y, 32)), 32) \text{ then } 0$
 $\quad \text{else } 1 \text{ endif}$

; the v-flag of asl. It is an easy copy of the definition of asl-v.
; sometimes, it seems to be better to define those functions directly
; using the intended meaning. But there are several problems:
; 1. the meaning is not completely clear. It is vague.

```

; 2. it is impossible to have a clean definition.
; 3. it is error prone. It does not always work the way you think.
;   Your intended interpretation is just a special case.
; In some sense, a formal specification should be more syntax-oriented.
; syntax rules are clear and accurate. e.g. M68000 are well-documented.
; But it gives one a hard time to do formal proofs. We need to assign
; meanings to these definitions. To make sure our "intended" meanings
; are consistent with the specification, we need to prove the equivalence.
; this is done formally by a theorem prover, and sometimes, is very hard.

```

THEOREM: asl-bvs

```

bvs (asl-v (n, x, cnt))
=   if int-rangep (nat-to-int (x, n), n - cnt) then 0
    else 1 endif

```

EVENT: Disable bvs.

```

;                                     BMI/BPL
; the n-flag of move.

```

THEOREM: move-bmi

```

nat-rangep (x, n)
→ (bmi (move-n (n, x))
   =   if negativep (nat-to-int (x, n)) then 1
       else 0 endif)

```

; the n-flag of sub.

THEOREM: sub-bmi

```

(nat-rangep (x, n) ∧ nat-rangep (y, n) ∧ (n ≠ 0))
→ (bmi (sub-n (n, x, y))
   =   if int-rangep (idifference (nat-to-int (y, n), nat-to-int (x, n)), n)
       then if illessp (nat-to-int (y, n), nat-to-int (x, n)) then 1
              else 0 endif
       elseif illessp (nat-to-int (y, n), nat-to-int (x, n)) then 0
       else 1 endif)

```

; the n-flag of add.

THEOREM: add-bmi

```

(nat-rangep (x, n) ∧ nat-rangep (y, n) ∧ (n ≠ 0))
→ (bmi (add-n (n, x, y))
   =   if int-rangep (iplus (nat-to-int (x, n), nat-to-int (y, n)), n)
       then if negativep (iplus (nat-to-int (x, n), nat-to-int (y, n)))
              then 1

```

```

        else 0 endif
    elseif negativep (nat-to-int (x, n)) then 0
    else 1 endif)

```

; the n-flag of muls. There are three cases to handle.

THEOREM: muls_1632-bmi
 (nat-rangep (x, 16) $\wedge$ nat-rangep (y, 16))
 $\rightarrow$ (bmi (muls-n (32, x, y, 16))
 = if (nat-to-int (x, 16) = 0) $\vee$ (nat-to-int (y, 16) = 0)
 then 0
 elseif negativep (nat-to-int (x, 16))
 then if negativep (nat-to-int (y, 16)) then 0
 else 1 endif
 elseif negativep (nat-to-int (y, 16)) then 1
 else 0 endif)

THEOREM: muls_3264-bmi
 (nat-rangep (x, 32) $\wedge$ nat-rangep (y, 32))
 $\rightarrow$ (bmi (muls-n (64, x, y, 32))
 = if (nat-to-int (x, 32) = 0) $\vee$ (nat-to-int (y, 32) = 0)
 then 0
 elseif negativep (nat-to-int (x, 32))
 then if negativep (nat-to-int (y, 32)) then 0
 else 1 endif
 elseif negativep (nat-to-int (y, 32)) then 1
 else 0 endif)

THEOREM: muls_3232-bmi
 (int-rangep (itimes (nat-to-int (x, 32), nat-to-int (y, 32)), 32)
 $\wedge$ nat-rangep (x, 32)
 $\wedge$ nat-rangep (y, 32))
 $\rightarrow$ (bmi (muls-n (32, x, y, 32))
 = if (nat-to-int (x, 32) = 0) $\vee$ (nat-to-int (y, 32) = 0)
 then 0
 elseif negativep (nat-to-int (x, 32))
 then if negativep (nat-to-int (y, 32)) then 0
 else 1 endif
 elseif negativep (nat-to-int (y, 32)) then 1
 else 0 endif)

; the n-flag of asl.

THEOREM: asl-bmi
 (nat-rangep (x, n) $\wedge$ int-rangep (nat-to-int (x, n), n - cnt))

$\rightarrow$ (bmi (asl-n (n, x, cnt))
 $=$ if negativep (nat-to-int (x, n)) then 1
 else 0 endif)

; the n-flag of asr.

THEOREM: asr-bmi

nat-range (x, n)
 $\rightarrow$ (bmi (asr-n (n, x, cnt))
 $=$ if negativep (nat-to-int (x, n)) then 1
 else 0 endif)

; the n-flag of ext.

THEOREM: ext-bmi

(nat-range (x, n) $\wedge$ ($n < size$))
 $\rightarrow$ (bmi (ext-n ($n, x, size$))
 $=$ if negativep (nat-to-int (x, n)) then 1
 else 0 endif)

; BGE/BLT

THEOREM: bge-v0

bge ($0, n$) = b-not (bmi (n))

THEOREM: blt-v0

blt ($0, n$) = bmi (n)

; bge of sub.

THEOREM: sub-bge

(nat-range (x, n) $\wedge$ nat-range (y, n) $\wedge$ ($n \neq 0$))
 $\rightarrow$ (bge (sub-v (n, x, y), sub-n (n, x, y))
 $=$ if ilssp (nat-to-int (y, n), nat-to-int (x, n)) then 0
 else 1 endif)

; blt of sub.

THEOREM: sub-blt

(nat-range (x, n) $\wedge$ nat-range (y, n) $\wedge$ ($n \neq 0$))
 $\rightarrow$ (blt (sub-v (n, x, y), sub-n (n, x, y))
 $=$ if ilssp (nat-to-int (y, n), nat-to-int (x, n)) then 1
 else 0 endif)

; the trivial relation between BGE and BLT.

THEOREM: blt-bge
 $\text{blt}(v, n) = \text{b-not}(\text{bge}(v, n))$

EVENT: Disable bge.

EVENT: Disable blt.

; BGT/BLE

THEOREM: sub-z-lal
 $(\text{nat-range}(x, n) \wedge \text{nat-range}(y, n))$
 $\rightarrow (\text{sub-z}(n, x, y)$
 $= \text{if } \text{nat-to-int}(x, n) = \text{nat-to-int}(y, n) \text{ then } 1$
 $\text{else } 0 \text{ endif})$

EVENT: Disable sub-z.

; bgt of sub.

THEOREM: sub-bgt
 $(\text{nat-range}(x, n) \wedge \text{nat-range}(y, n) \wedge (n \neq 0))$
 $\rightarrow (\text{bgt}(\text{sub-v}(n, x, y), \text{sub-z}(n, x, y), \text{sub-n}(n, x, y))$
 $= \text{if } \text{ilessp}(\text{nat-to-int}(x, n), \text{nat-to-int}(y, n)) \text{ then } 1$
 $\text{else } 0 \text{ endif})$

; ble of sub.

THEOREM: sub-ble
 $(\text{nat-range}(x, n) \wedge \text{nat-range}(y, n) \wedge (n \neq 0))$
 $\rightarrow (\text{ble}(\text{sub-v}(n, x, y), \text{sub-z}(n, x, y), \text{sub-n}(n, x, y))$
 $= \text{if } \text{ilessp}(\text{nat-to-int}(x, n), \text{nat-to-int}(y, n)) \text{ then } 0$
 $\text{else } 1 \text{ endif})$

; bgt of move.

THEOREM: move-bgt
 $\text{nat-range}(x, n)$
 $\rightarrow (\text{bgt}(0, \text{move-z}(n, x), \text{move-n}(n, x))$
 $= \text{if } 0 < \text{nat-to-int}(x, n) \text{ then } 1$
 $\text{else } 0 \text{ endif})$

; ble of move.

THEOREM: move-ble
 nat-range $p(x, n)$
 $\rightarrow$ (ble(0, move-z(n, x), move-n(n, x))
 = **if** 0 < nat-to-int(x, n) **then** 0
 else 1 **endif**)

 ; the trivial relation between ble and bgt.

THEOREM: ble-bgt
 ble(v, z, n) = b-not(bgt(v, z, n))

EVENT: Disable ble.

EVENT: Disable bgt.

; the end of this subsection. Some lemmas should be disabled, for we
 ; no longer need them.

EVENT: Disable mbit-means-negativep.

EVENT: Disable bmi.

EVENT: Disable add-addx-c.

EVENT: Disable sub-subx-c.

EVENT: Disable addx-c-la.

EVENT: Disable subx-c-la.

EVENT: Disable add-addx-v.

EVENT: Disable subx-addx-v.

EVENT: Disable sub-subx-v.

EVENT: Disable addx-v-la.

EVENT: Disable subx-v-la.

EVENT: Disable sub-z-la.

EVENT: Disable sub-z-la1.

EVENT: Disable illessp-crock1.

EVENT: Disable illessp-crock2.

EVENT: Disable add-bmi-crock1.

EVENT: Disable add-bmi-crock2.

EVENT: Disable nat-to-int=.

;
; STEP
; stepping the state s $(m+n)$ steps is equivalent to stepping s m steps and
; n steps further.

DEFINITION: $\text{splus}(x, y) = (x + y)$

THEOREM: stepn-lemma
 $\text{stepn}(s, \text{splus}(m, n)) = \text{stepn}(\text{stepn}(s, m), n)$

THEOREM: stepn-rewriter0
 $\text{stepn}(s, 0) = s$

THEOREM: stepn-rewriter
 $(\text{mc-status}(s) = \text{'running'}) \rightarrow (\text{stepn}(s, 1 + n) = \text{stepn}(\text{stepi}(s), n))$

;
; THEOREMS ABOUT THE MEMORY
; if a portion of memory is ROM, then it still is ROM after a write on memory.

THEOREM: byte-write-maintain-rom-addrp
 $\text{rom-addrp}(\text{addr}, \text{byte-write}(\text{value}, x, \text{mem}), n) = \text{rom-addrp}(\text{addr}, \text{mem}, n)$

THEOREM: write-mem-maintain-rom-addrp
 $\text{rom-addrp}(\text{addr}, \text{write-mem}(\text{value}, x, \text{mem}, m), n) = \text{rom-addrp}(\text{addr}, \text{mem}, n)$

; if a portion of memory is RAM, then it still is RAM after a write on memory.

THEOREM: byte-write-maintain-ram-addrp
 $\text{ram-addrp}(\text{addr}, \text{byte-write}(\text{value}, x, \text{mem}), n) = \text{ram-addrp}(\text{addr}, \text{mem}, n)$

THEOREM: write-mem-maintain-ram-addrp

$\text{ram-addrp}(\text{addr}, \text{write-mem}(\text{value}, x, \text{mem}, m), n) = \text{ram-addrp}(\text{addr}, \text{mem}, n)$

; mcode-addrp claims that the machine code program *lst* is stored in the
; memory starting at *addr*.

DEFINITION:

$\text{mcode-addrp}(\text{addr}, \text{mem}, \text{lst})$

= **if** $\text{listp}(\text{lst})$

then if $\text{car}(\text{lst}) = -1$

then $\text{mcode-addrp}(\text{add}(32, \text{addr}, 1), \text{mem}, \text{cdr}(\text{lst}))$

else $(\text{pc-byte-read}(\text{addr}, \text{mem}) = \text{car}(\text{lst}))$

$\wedge \text{mcode-addrp}(\text{add}(32, \text{addr}, 1), \text{mem}, \text{cdr}(\text{lst}))$ **endif**

else t endif

THEOREM: add-non-numberp

$(i \notin \mathbf{N}) \rightarrow (\text{add}(n, x, i) = \text{head}(x, n))$

THEOREM: add-plus

$\text{add}(n, x, \text{add}(n, y, z)) = \text{add}(n, x, y + z)$

; four lemmas about pc-read-memp.

THEOREM: pc-read-memp-la0

$(\text{pc-read-memp}(\text{addr}, \text{mem}, k) \wedge (k \neq 0)) \rightarrow \text{pc-byte-readp}(\text{addr}, \text{mem})$

THEOREM: pc-read-memp-la1

$(\text{pc-read-memp}(\text{addr}, \text{mem}, k) \wedge (j < k))$

$\rightarrow \text{pc-byte-readp}(\text{add}(32, \text{addr}, j), \text{mem})$

THEOREM: pc-read-memp-la2

$(\text{pc-read-memp}(\text{addr}, \text{mem}, k) \wedge (j \leq k)) \rightarrow \text{pc-read-memp}(\text{addr}, \text{mem}, j)$

THEOREM: pc-read-memp-la3

$(\text{pc-read-memp}(\text{addr}, \text{mem}, k) \wedge ((i + j) \leq k))$

$\rightarrow \text{pc-read-memp}(\text{add}(32, \text{addr}, i), \text{mem}, j)$

THEOREM: write-memp-la0

$(\text{write-memp}(\text{addr}, \text{mem}, n) \wedge (n \neq 0)) \rightarrow \text{byte-writep}(\text{addr}, \text{mem})$

THEOREM: write-memp-la1

$(\text{write-memp}(\text{addr}, \text{mem}, n) \wedge (m < n)) \rightarrow \text{byte-writep}(\text{add}(32, \text{addr}, m), \text{mem})$

THEOREM: write-memp-la2

$(\text{write-memp}(\text{addr}, \text{mem}, n) \wedge (m \leq n)) \rightarrow \text{write-memp}(\text{addr}, \text{mem}, m)$

THEOREM: write-memp-la3
 $(\text{write-memp}(addr, mem, n) \wedge ((i + j) \leq n))$
 $\rightarrow \text{write-memp}(\text{add}(32, addr, i), mem, j)$

EVENT: Disable add-plus.

; array index. There are some good reasons to distinguish this concept
 ; with add and sub.

DEFINITION: $\text{index-n}(x, y) = \text{sub}(32, y, x)$

; ROM is pc-readable.

THEOREM: pc-byte-readp-rom0
 $(\text{rom-addrp}(addr, mem, k) \wedge (k \neq 0)) \rightarrow \text{pc-byte-readp}(addr, mem)$

THEOREM: pc-byte-readp-rom1
 $(\text{rom-addrp}(addr, mem, k) \wedge (j < k))$
 $\rightarrow \text{pc-byte-readp}(\text{add}(32, addr, j), mem)$

THEOREM: pc-byte-readp-rom2
 $(\text{rom-addrp}(\text{add}(32, addr, i), mem, k) \wedge (\text{index-n}(0, i) < k))$
 $\rightarrow \text{pc-byte-readp}(addr, mem)$

THEOREM: pc-byte-readp-rom3
 $(\text{rom-addrp}(\text{add}(32, addr, i), mem, k) \wedge (\text{index-n}(j, i) < k))$
 $\rightarrow \text{pc-byte-readp}(\text{add}(32, addr, j), mem)$

THEOREM: pc-read-memp-rom0
 $(\text{rom-addrp}(addr, mem, k) \wedge (j \leq k)) \rightarrow \text{pc-read-memp}(addr, mem, j)$

THEOREM: pc-read-memp-rom1
 $(\text{rom-addrp}(addr, mem, k) \wedge ((i + j) \leq k))$
 $\rightarrow \text{pc-read-memp}(\text{add}(32, addr, i), mem, j)$

THEOREM: pc-read-memp-rom2
 $(\text{rom-addrp}(\text{add}(32, addr, i), mem, k) \wedge ((\text{index-n}(0, i) + j) \leq k))$
 $\rightarrow \text{pc-read-memp}(addr, mem, j)$

THEOREM: pc-read-memp-rom3
 $(\text{rom-addrp}(\text{add}(32, addr, i), mem, k) \wedge ((\text{index-n}(h, i) + j) \leq k))$
 $\rightarrow \text{pc-read-memp}(\text{add}(32, addr, h), mem, j)$

; starting at the same address, a portion of the memory is ROM if a bigger
 ; portion of the memory is ROM.

THEOREM: rom-addrp-la1

$(\text{rom-addrp}(addr, mem, n) \wedge (m \leq n)) \rightarrow \text{rom-addrp}(addr, mem, m)$

; a portion of the memory is also ROM, if a bigger portion of the memory
; is ROM.

THEOREM: rom-addrp-la2

$(\text{rom-addrp}(addr, mem, k) \wedge ((j + \text{index-n}(x, addr)) \leq k))$
 $\rightarrow \text{rom-addrp}(x, mem, j)$

EVENT: Disable rom-addrp.

; ROM is readable.

THEOREM: byte-readp-rom0

$(\text{rom-addrp}(addr, mem, k) \wedge (k \neq 0)) \rightarrow \text{byte-readp}(addr, mem)$

THEOREM: byte-readp-rom1

$(\text{rom-addrp}(addr, mem, k) \wedge (j < k)) \rightarrow \text{byte-readp}(\text{add}(32, addr, j), mem)$

THEOREM: byte-readp-rom2

$(\text{rom-addrp}(\text{add}(32, addr, i), mem, k) \wedge (\text{index-n}(0, i) < k))$
 $\rightarrow \text{byte-readp}(addr, mem)$

THEOREM: byte-readp-rom3

$(\text{rom-addrp}(\text{add}(32, addr, i), mem, k) \wedge (\text{index-n}(j, i) < k))$
 $\rightarrow \text{byte-readp}(\text{add}(32, addr, j), mem)$

THEOREM: read-memp-rom0

$(\text{rom-addrp}(addr, mem, k) \wedge (j \leq k)) \rightarrow \text{read-memp}(addr, mem, j)$

THEOREM: read-memp-rom1

$(\text{rom-addrp}(addr, mem, k) \wedge ((i + j) \leq k))$
 $\rightarrow \text{read-memp}(\text{add}(32, addr, i), mem, j)$

THEOREM: read-memp-rom2

$(\text{rom-addrp}(\text{add}(32, addr, i), mem, k) \wedge ((\text{index-n}(0, i) + j) \leq k))$
 $\rightarrow \text{read-memp}(addr, mem, j)$

THEOREM: read-memp-rom3

$(\text{rom-addrp}(\text{add}(32, addr, i), mem, k) \wedge ((\text{index-n}(h, i) + j) \leq k))$
 $\rightarrow \text{read-memp}(\text{add}(32, addr, h), mem, j)$

; integer view.

THEOREM: read-memp-rom1-int

$(\text{rom-addrp}(addr, mem, n)$
 $\wedge ((\text{nat-to-int}(x, 32) + j) < n)$
 $\wedge (\text{nat-to-int}(x, 32) \in \mathbf{N}))$
 $\rightarrow \text{read-memp}(\text{add}(32, addr, x), mem, j)$

; RAM is writable.

THEOREM: byte-writep-ram0

$(\text{ram-addrp}(addr, mem, k) \wedge (k \neq 0)) \rightarrow \text{byte-writep}(addr, mem)$

THEOREM: byte-writep-ram1

$(\text{ram-addrp}(addr, mem, k) \wedge (j < k)) \rightarrow \text{byte-writep}(\text{add}(32, addr, j), mem)$

THEOREM: byte-writep-ram2

$(\text{ram-addrp}(\text{add}(32, addr, i), mem, k) \wedge (\text{index-n}(0, i) < k))$
 $\rightarrow \text{byte-writep}(addr, mem)$

THEOREM: byte-writep-ram3

$(\text{ram-addrp}(\text{add}(32, addr, i), mem, k) \wedge (\text{index-n}(j, i) < k))$
 $\rightarrow \text{byte-writep}(\text{add}(32, addr, j), mem)$

THEOREM: write-memp-ram0

$(\text{ram-addrp}(addr, mem, k) \wedge (j \leq k)) \rightarrow \text{write-memp}(addr, mem, j)$

THEOREM: write-memp-ram1

$(\text{ram-addrp}(addr, mem, k) \wedge ((i + j) \leq k))$
 $\rightarrow \text{write-memp}(\text{add}(32, addr, i), mem, j)$

THEOREM: write-memp-ram1-int

$(\text{ram-addrp}(addr, mem, k)$
 $\wedge (\text{nat-to-int}(i, 32) \in \mathbf{N})$
 $\wedge ((\text{nat-to-int}(i, 32) + j) \leq k))$
 $\rightarrow \text{write-memp}(\text{add}(32, addr, i), mem, j)$

THEOREM: write-memp-ram2

$(\text{ram-addrp}(\text{add}(32, addr, i), mem, k) \wedge ((\text{index-n}(0, i) + j) \leq k))$
 $\rightarrow \text{write-memp}(addr, mem, j)$

THEOREM: write-memp-ram3

$(\text{ram-addrp}(\text{add}(32, addr, i), mem, k) \wedge ((\text{index-n}(h, i) + j) \leq k))$
 $\rightarrow \text{write-memp}(\text{add}(32, addr, h), mem, j)$

; a portion of a memory is also RAM, if a "bigger" portion of the memory
; is RAM.

THEOREM: ram-addrp-la1
 $(\text{ram-addrp}(addr, mem, n) \wedge (m \leq n)) \rightarrow \text{ram-addrp}(addr, mem, m)$

THEOREM: ram-addrp-la2
 $(\text{ram-addrp}(addr, mem, n) \wedge ((j + \text{index-n}(x, addr)) \leq n))$
 $\rightarrow \text{ram-addrp}(x, mem, j)$

EVENT: Disable ram-addrp.

; RAM is readable.

THEOREM: byte-readp-ram0
 $(\text{ram-addrp}(addr, mem, k) \wedge (k \neq 0)) \rightarrow \text{byte-readp}(addr, mem)$

THEOREM: byte-readp-ram1
 $(\text{ram-addrp}(addr, mem, k) \wedge (j < k)) \rightarrow \text{byte-readp}(\text{add}(32, addr, j), mem)$

THEOREM: byte-readp-ram2
 $(\text{ram-addrp}(\text{add}(32, addr, i), mem, k) \wedge (\text{index-n}(0, i) < k))$
 $\rightarrow \text{byte-readp}(addr, mem)$

THEOREM: byte-readp-ram3
 $(\text{ram-addrp}(\text{add}(32, addr, i), mem, k) \wedge (\text{index-n}(j, i) < k))$
 $\rightarrow \text{byte-readp}(\text{add}(32, addr, j), mem)$

THEOREM: read-memp-ram0
 $(\text{ram-addrp}(addr, mem, k) \wedge (j \leq k)) \rightarrow \text{read-memp}(addr, mem, j)$

THEOREM: read-memp-ram1
 $(\text{ram-addrp}(addr, mem, k) \wedge ((i + j) \leq k))$
 $\rightarrow \text{read-memp}(\text{add}(32, addr, i), mem, j)$

THEOREM: read-memp-ram1-int
 $(\text{ram-addrp}(addr, mem, k)$
 $\wedge (\text{nat-to-int}(i, 32) \in \mathbf{N})$
 $\wedge ((\text{nat-to-int}(i, 32) + j) \leq k))$
 $\rightarrow \text{read-memp}(\text{add}(32, addr, i), mem, j)$

THEOREM: read-memp-ram2
 $(\text{ram-addrp}(\text{add}(32, addr, i), mem, k) \wedge ((\text{index-n}(0, i) + j) \leq k))$
 $\rightarrow \text{read-memp}(addr, mem, j)$

THEOREM: read-memp-ram3
 $(\text{ram-addrp}(\text{add}(32, addr, i), mem, k) \wedge ((\text{index-n}(h, i) + j) \leq k))$
 $\rightarrow \text{read-memp}(\text{add}(32, addr, h), mem, j)$

; please use the right induction.

DEFINITION:

mem-induct1(i , $addr$, lst)
= **if** $i \simeq 0$ **then** t
 else mem-induct1($i - 1$, add(32, $addr$, 1), cdr(lst)) **endif**

; OBTAIN MACHINE CODE FROM MEMORY
; please read the right thing.

THEOREM: pc-byte-read-mcode0
(mcode-addrp($addr$, mem , cons(x , lst)) $\wedge$ ($x \neq -1$))
→ (pc-byte-read($addr$, mem) = x)

THEOREM: pc-byte-read-mcode1
(mcode-addrp($addr$, mem , lst) $\wedge$ ($i < \text{len}(lst)$) $\wedge$ (get-nth(i , lst) $\neq -1$))
→ (pc-byte-read(add(32, $addr$, i), mem) = get-nth(i , lst))

THEOREM: pc-byte-read-mcode2
(mcode-addrp(add(32, $addr$, i), mem , lst)
 $\wedge$ (index-n(0, i) < len(lst))
 $\wedge$ (get-nth(index-n(0, i), lst) $\neq -1$))
→ (pc-byte-read($addr$, mem) = get-nth(index-n(0, i), lst))

THEOREM: pc-byte-read-mcode3
(mcode-addrp(add(32, $addr$, i), mem , lst)
 $\wedge$ (index-n(j , i) < len(lst))
 $\wedge$ (get-nth(index-n(j , i), lst) $\neq -1$))
→ (pc-byte-read(add(32, $addr$, j), mem) = get-nth(index-n(j , i), lst))

DEFINITION:

lst-numberp0(lst , n)
= **if** $n \simeq 0$ **then** 0
 else (get-nth($n - 1$, lst) $\in \mathbf{N}$) $\wedge$ lst-numberp0(lst , $n - 1$) **endif**

DEFINITION:

read-lst0(lst , n)
= **if** $n \simeq 0$ **then** 0
 else app(B, get-nth($n - 1$, lst), read-lst0(lst , $n - 1$)) **endif**

DEFINITION:

tail-lst(lst , n)
= **if** $n \simeq 0$ **then** lst
 else tail-lst(cdr(lst), $n - 1$) **endif**

DEFINITION:

lst-numberp(m , lst , n) = lst-numberp0(tail-lst(lst , m), n)

DEFINITION: $\text{read-lst}(m, \text{lst}, n) = \text{read-lst0}(\text{tail-lst}(\text{lst}, m), n)$

THEOREM: get-nth-tail-lst
 $\text{get-nth}(n, \text{tail-lst}(x, m)) = \text{get-nth}(m + n, x)$

THEOREM: pc-read-mem-mcode0
 $(\text{mcode-addrp}(\text{addr}, \text{mem}, \text{lst}) \wedge (j \leq \text{len}(\text{lst})) \wedge \text{lst-numberp}(0, \text{lst}, j))$
 $\rightarrow (\text{pc-read-mem}(\text{addr}, \text{mem}, j) = \text{read-lst}(0, \text{lst}, j))$

THEOREM: pc-read-mem-mcode1
 $(\text{mcode-addrp}(\text{addr}, \text{mem}, \text{lst})$
 $\wedge ((i + j) \leq \text{len}(\text{lst}))$
 $\wedge \text{lst-numberp}(i, \text{lst}, j))$
 $\rightarrow (\text{pc-read-mem}(\text{add}(32, \text{addr}, i), \text{mem}, j) = \text{read-lst}(i, \text{lst}, j))$

EVENT: Disable read-lst.

EVENT: Disable lst-numberp.

THEOREM: pc-read-mem-mcode2
 $(\text{mcode-addrp}(\text{add}(32, \text{addr}, i), \text{mem}, \text{lst})$
 $\wedge ((\text{index-n}(0, i) + k) \leq \text{len}(\text{lst}))$
 $\wedge \text{lst-numberp}(\text{index-n}(0, i), \text{lst}, k))$
 $\rightarrow (\text{pc-read-mem}(\text{addr}, \text{mem}, k) = \text{read-lst}(\text{index-n}(0, i), \text{lst}, k))$

THEOREM: pc-read-mem-mcode3
 $(\text{mcode-addrp}(\text{add}(32, \text{addr}, i), \text{mem}, \text{lst})$
 $\wedge ((\text{index-n}(j, i) + k) \leq \text{len}(\text{lst}))$
 $\wedge \text{lst-numberp}(\text{index-n}(j, i), \text{lst}, k))$
 $\rightarrow (\text{pc-read-mem}(\text{add}(32, \text{addr}, j), \text{mem}, k) = \text{read-lst}(\text{index-n}(j, i), \text{lst}, k))$

; sometimes, we obtain machine code by read-mem.

THEOREM: read->pc-read-mem
 $\text{read-mem}(x, \text{mem}, k) = \text{pc-read-mem}(x, \text{mem}, k)$

THEOREM: read-mem-mcode1-int
 $(\text{mcode-addrp}(\text{addr}, \text{mem}, \text{lst})$
 $\wedge ((\text{nat-to-int}(i, 32) + j) \leq \text{len}(\text{lst}))$
 $\wedge (\text{nat-to-int}(i, 32) \in \mathbf{N})$
 $\wedge \text{lst-numberp}(\text{nat-to-int}(i, 32), \text{lst}, j))$
 $\rightarrow (\text{read-mem}(\text{add}(32, \text{addr}, i), \text{mem}, j)$
 $= \text{read-lst}(\text{nat-to-int}(i, 32), \text{lst}, j))$

THEOREM: read-mem-mcode2

$$\begin{aligned}
& (\text{mcode-addrp}(\text{add}(32, \text{addr}, i), \text{mem}, \text{lst}) \\
& \quad \wedge ((\text{index-n}(0, i) + k) \leq \text{len}(\text{lst})) \\
& \quad \wedge \text{lst-numberp}(\text{index-n}(0, i), \text{lst}, k)) \\
& \rightarrow (\text{read-mem}(\text{addr}, \text{mem}, k) = \text{read-lst}(\text{index-n}(0, i), \text{lst}, k))
\end{aligned}$$

THEOREM: read-mem-mcode3

$$\begin{aligned}
& (\text{mcode-addrp}(\text{add}(32, \text{addr}, i), \text{mem}, \text{lst}) \\
& \quad \wedge ((\text{index-n}(j, i) + k) \leq \text{len}(\text{lst})) \\
& \quad \wedge \text{lst-numberp}(\text{index-n}(j, i), \text{lst}, k)) \\
& \rightarrow (\text{read-mem}(\text{add}(32, \text{addr}, j), \text{mem}, k) = \text{read-lst}(\text{index-n}(j, i), \text{lst}, k))
\end{aligned}$$

EVENT: Disable read->pc-read-mem.

; program segment remains unchanged after any write on memory, because
; program is in ROM.

THEOREM: byte-write-mcode-addrp

$$\begin{aligned}
& (\text{pc-read-memp}(\text{pc}, \text{mem}, \text{len}(\text{lst})) \wedge \text{byte-writep}(x, \text{mem})) \\
& \rightarrow (\text{mcode-addrp}(\text{pc}, \text{byte-write}(\text{value}, x, \text{mem}), \text{lst}) \\
& \quad = \text{mcode-addrp}(\text{pc}, \text{mem}, \text{lst}))
\end{aligned}$$

THEOREM: write-mem-mcode-addrp

$$\begin{aligned}
& (\text{pc-read-memp}(\text{pc}, \text{mem}, \text{len}(\text{lst})) \wedge \text{write-memp}(x, \text{mem}, n)) \\
& \rightarrow (\text{mcode-addrp}(\text{pc}, \text{write-mem}(\text{value}, x, \text{mem}, n), \text{lst}) \\
& \quad = \text{mcode-addrp}(\text{pc}, \text{mem}, \text{lst}))
\end{aligned}$$

; DISJOINT0 AND DISJOINT

THEOREM: disjoint0-head

$$\begin{aligned}
& (\text{disjoint0}(\text{head}(x, 32), m, y) = \text{disjoint0}(x, m, y)) \\
& \wedge (\text{disjoint0}(x, m, \text{head}(y, 32)) = \text{disjoint0}(x, m, y))
\end{aligned}$$

THEOREM: disjoint-head

$$\begin{aligned}
& (\text{disjoint}(\text{head}(x, 32), m, y, n) = \text{disjoint}(x, m, y, n)) \\
& \wedge (\text{disjoint}(x, m, \text{head}(y, 32), n) = \text{disjoint}(x, m, y, n))
\end{aligned}$$

THEOREM: disjoint0-la0

$$(\text{disjoint0}(a, m, b) \wedge (i < m)) \rightarrow (\neg \text{mod32-eq}(\text{add}(32, a, i), b))$$

THEOREM: disjoint0-la1

$$(\text{disjoint0}(a, m, b) \wedge (i < m)) \rightarrow \text{disjoint0}(a, i, b)$$

THEOREM: disjoint0-la2

$$(\text{disjoint0}(a, m, b) \wedge ((i + j) \leq m)) \rightarrow \text{disjoint0}(\text{add}(32, a, i), j, b)$$

THEOREM: disjoint-la0

$$(\text{disjoint}(a, m, b, n) \wedge (j < n)) \rightarrow \text{disjoint0}(a, m, \text{add}(32, b, j))$$

THEOREM: disjoint-la1

$$\begin{aligned} &(\text{disjoint}(a, m, b, n) \wedge (i < m) \wedge (j < n)) \\ \rightarrow &(\neg \text{mod32-eq}(\text{add}(32, a, i), \text{add}(32, b, j))) \end{aligned}$$

THEOREM: disjoint-la2

$$\begin{aligned} &(\text{disjoint}(a, m, b, n) \wedge ((i + j) \leq m) \wedge (k < n)) \\ \rightarrow &\text{disjoint0}(\text{add}(32, a, i), j, \text{add}(32, b, k)) \end{aligned}$$

THEOREM: disjoint-la3

$$\begin{aligned} &(\text{disjoint}(a, m, b, n) \wedge ((i + j) \leq m) \wedge ((k + l) \leq n)) \\ \rightarrow &\text{disjoint}(\text{add}(32, a, i), j, \text{add}(32, b, k), l) \end{aligned}$$

;

THEOREM: disjoint-0

$$(\text{disjoint}(a, m, b, n) \wedge (j \leq m) \wedge (l \leq n)) \rightarrow \text{disjoint}(a, j, b, l)$$

THEOREM: disjoint-1

$$\begin{aligned} &(\text{disjoint}(a, m, b, n) \wedge (j \leq m) \wedge ((k + l) \leq n)) \\ \rightarrow &\text{disjoint}(a, j, \text{add}(32, b, k), l) \end{aligned}$$

THEOREM: disjoint-2

$$\begin{aligned} &(\text{disjoint}(a, m, b, n) \wedge ((i + j) \leq m) \wedge (l \leq n)) \\ \rightarrow &\text{disjoint}(\text{add}(32, a, i), j, b, l) \end{aligned}$$

THEOREM: disjoint-3

$$\begin{aligned} &(\text{disjoint}(a, m, b, n) \wedge ((i + j) \leq m) \wedge ((k + l) \leq n)) \\ \rightarrow &\text{disjoint}(\text{add}(32, a, i), j, \text{add}(32, b, k), l) \end{aligned}$$

;

THEOREM: disjoint-4

$$\begin{aligned} &(\text{disjoint}(\text{add}(32, a, i), m, b, n) \wedge ((j + \text{index-n}(0, i)) \leq m) \wedge (l \leq n)) \\ \rightarrow &\text{disjoint}(a, j, b, l) \end{aligned}$$

THEOREM: disjoint-5

$$\begin{aligned} &(\text{disjoint}(\text{add}(32, a, i), m, b, n) \\ &\wedge ((j + \text{index-n}(0, i)) \leq m) \\ &\wedge ((k + l) \leq n)) \\ \rightarrow &\text{disjoint}(a, j, \text{add}(32, b, k), l) \end{aligned}$$

THEOREM: disjoint-6

$$\begin{aligned} &(\text{disjoint}(\text{add}(32, a, i), m, b, n) \\ &\wedge ((j + \text{index-n}(i1, i)) \leq m) \\ &\wedge (l \leq n)) \\ \rightarrow &\text{disjoint}(\text{add}(32, a, i1), j, b, l) \end{aligned}$$

THEOREM: disjoint-7

$$\begin{aligned} & (\text{disjoint}(\text{add}(32, a, i), m, b, n) \\ & \quad \wedge ((j + \text{index-n}(i1, i)) \leq m) \\ & \quad \wedge ((k + l) \leq n)) \\ & \rightarrow \text{disjoint}(\text{add}(32, a, i1), j, \text{add}(32, b, k), l) \end{aligned}$$

;

THEOREM: disjoint-8

$$\begin{aligned} & (\text{disjoint}(a, m, \text{add}(32, b, k), n) \wedge (j \leq m) \wedge ((l + \text{index-n}(0, k)) \leq n)) \\ & \rightarrow \text{disjoint}(a, j, b, l) \end{aligned}$$

THEOREM: disjoint-9

$$\begin{aligned} & (\text{disjoint}(a, m, \text{add}(32, b, k), n) \\ & \quad \wedge ((i + j) \leq m) \\ & \quad \wedge ((l + \text{index-n}(0, k)) \leq n)) \\ & \rightarrow \text{disjoint}(\text{add}(32, a, i), j, b, l) \end{aligned}$$

THEOREM: disjoint-10

$$\begin{aligned} & (\text{disjoint}(a, m, \text{add}(32, b, k), n) \\ & \quad \wedge (j \leq m) \\ & \quad \wedge ((l + \text{index-n}(k1, k)) \leq n)) \\ & \rightarrow \text{disjoint}(a, j, \text{add}(32, b, k1), l) \end{aligned}$$

THEOREM: disjoint-11

$$\begin{aligned} & (\text{disjoint}(a, m, \text{add}(32, b, k), n) \\ & \quad \wedge ((i + j) \leq m) \\ & \quad \wedge ((l + \text{index-n}(k1, k)) \leq n)) \\ & \rightarrow \text{disjoint}(\text{add}(32, a, i), j, \text{add}(32, b, k1), l) \end{aligned}$$

;

THEOREM: disjoint-12

$$\begin{aligned} & (\text{disjoint}(\text{add}(32, a, i), m, \text{add}(32, b, k), n) \\ & \quad \wedge ((j + \text{index-n}(0, i)) \leq m) \\ & \quad \wedge ((l + \text{index-n}(0, k)) \leq n)) \\ & \rightarrow \text{disjoint}(a, j, b, l) \end{aligned}$$

THEOREM: disjoint-13

$$\begin{aligned} & (\text{disjoint}(\text{add}(32, a, i), m, \text{add}(32, b, k), n) \\ & \quad \wedge ((j + \text{index-n}(i1, i)) \leq m) \\ & \quad \wedge ((l + \text{index-n}(0, k)) \leq n)) \\ & \rightarrow \text{disjoint}(\text{add}(32, a, i1), j, b, l) \end{aligned}$$

THEOREM: disjoint-14

$$\begin{aligned}
& (\text{disjoint}(\text{add}(32, a, i), m, \text{add}(32, b, k), n) \\
& \quad \wedge ((j + \text{index-n}(0, i)) \leq m) \\
& \quad \wedge ((l + \text{index-n}(k1, k)) \leq n)) \\
& \rightarrow \text{disjoint}(a, j, \text{add}(32, b, k1), l)
\end{aligned}$$

THEOREM: disjoint-15

$$\begin{aligned}
& (\text{disjoint}(\text{add}(32, a, i), m, \text{add}(32, b, k), n) \\
& \quad \wedge ((j + \text{index-n}(i1, i)) \leq m) \\
& \quad \wedge ((l + \text{index-n}(k1, k)) \leq n)) \\
& \rightarrow \text{disjoint}(\text{add}(32, a, i1), j, \text{add}(32, b, k1), l)
\end{aligned}$$

; the commutativity of disjoint.

THEOREM: disjoint-commutativity

$$\text{disjoint}(x, m, y, n) = \text{disjoint}(y, n, x, m)$$

; the dual events of the above 16 events.

THEOREM: disjoint-0~ $(\text{disjoint}(a, m, b, n) \wedge (j \leq m) \wedge (l \leq n)) \rightarrow \text{disjoint}(b, l, a, j)$

THEOREM: disjoint-1~ $(\text{disjoint}(a, m, b, n) \wedge (j \leq m) \wedge ((k + l) \leq n))$
 $\rightarrow \text{disjoint}(\text{add}(32, b, k), l, a, j)$

THEOREM: disjoint-2~ $(\text{disjoint}(a, m, b, n) \wedge ((i + j) \leq m) \wedge (l \leq n))$
 $\rightarrow \text{disjoint}(b, l, \text{add}(32, a, i), j)$

THEOREM: disjoint-3~ $(\text{disjoint}(a, m, b, n) \wedge ((i + j) \leq m) \wedge ((k + l) \leq n))$
 $\rightarrow \text{disjoint}(\text{add}(32, b, k), l, \text{add}(32, a, i), j)$

;

THEOREM: disjoint-4~ $(\text{disjoint}(\text{add}(32, a, i), m, b, n) \wedge ((j + \text{index-n}(0, i)) \leq m) \wedge (l \leq n))$
 $\rightarrow \text{disjoint}(b, l, a, j)$

THEOREM: disjoint-5~ $(\text{disjoint}(\text{add}(32, a, i), m, b, n)$
 $\quad \wedge ((j + \text{index-n}(0, i)) \leq m)$
 $\quad \wedge ((k + l) \leq n))$
 $\rightarrow \text{disjoint}(\text{add}(32, b, k), l, a, j)$

THEOREM: disjoint-6~ $(\text{disjoint}(\text{add}(32, a, i), m, b, n)$
 $\quad \wedge ((j + \text{index-n}(i1, i)) \leq m)$
 $\quad \wedge (l \leq n))$
 $\rightarrow \text{disjoint}(b, l, \text{add}(32, a, i1), j)$

THEOREM: disjoint-7~ (disjoint (add (32, a, i), m, b, n)
 $\wedge ((j + \text{index-n}(i1, i)) \leq m)$
 $\wedge ((k + l) \leq n))$
 $\rightarrow$ disjoint (add (32, b, k), l, add (32, a, i1), j)

;

THEOREM: disjoint-8~ (disjoint (a, m, add (32, b, k), n) $\wedge (j \leq m) \wedge ((l + \text{index-n}(0, k)) \leq n))$
 $\rightarrow$ disjoint (b, l, a, j)

THEOREM: disjoint-9~ (disjoint (a, m, add (32, b, k), n)
 $\wedge ((i + j) \leq m)$
 $\wedge ((l + \text{index-n}(0, k)) \leq n))$
 $\rightarrow$ disjoint (b, l, add (32, a, i), j)

THEOREM: disjoint-10~ (disjoint (a, m, add (32, b, k), n)
 $\wedge (j \leq m)$
 $\wedge ((l + \text{index-n}(k1, k)) \leq n))$
 $\rightarrow$ disjoint (add (32, b, k1), l, a, j)

THEOREM: disjoint-11~ (disjoint (a, m, add (32, b, k), n)
 $\wedge ((i + j) \leq m)$
 $\wedge ((l + \text{index-n}(k1, k)) \leq n))$
 $\rightarrow$ disjoint (add (32, b, k1), l, add (32, a, i), j)

;

THEOREM: disjoint-12~ (disjoint (add (32, a, i), m, add (32, b, k), n)
 $\wedge ((j + \text{index-n}(0, i)) \leq m)$
 $\wedge ((l + \text{index-n}(0, k)) \leq n))$
 $\rightarrow$ disjoint (b, l, a, j)

THEOREM: disjoint-13~ (disjoint (add (32, a, i), m, add (32, b, k), n)
 $\wedge ((j + \text{index-n}(i1, i)) \leq m)$
 $\wedge ((l + \text{index-n}(0, k)) \leq n))$
 $\rightarrow$ disjoint (b, l, add (32, a, i1), j)

THEOREM: disjoint-14~ (disjoint (add (32, a, i), m, add (32, b, k), n)
 $\wedge ((j + \text{index-n}(0, i)) \leq m)$
 $\wedge ((l + \text{index-n}(k1, k)) \leq n))$
 $\rightarrow$ disjoint (add (32, b, k1), l, a, j)

THEOREM: disjoint-15~ (disjoint (add (32, a, i), m, add (32, b, k), n)
 $\wedge ((j + \text{index-n}(i1, i)) \leq m)$
 $\wedge ((l + \text{index-n}(k1, k)) \leq n))$
 $\rightarrow$ disjoint (add (32, b, k1), l, add (32, a, i1), j)

; disjoint with asl.

THEOREM: times-plus-lessp-cancel

$$\begin{aligned} & ((a \leq k) \wedge (b \leq k) \wedge (a < b)) \\ \rightarrow & (((a + (i * k)) < (b + (j * k))) = (i \leq j)) \end{aligned}$$

THEOREM: disjoint-2-asl

$$\begin{aligned} & (\text{disjoint } (a, m, b, n) \\ & \wedge \quad (\text{opsz} = \exp(2, cnt)) \\ & \wedge \quad (\text{nat-to-int } (x, 32) < (m \div \text{opsz})) \\ & \wedge \quad (\text{nat-to-int } (x, 32) \in \mathbf{N}) \\ & \wedge \quad (l \leq n)) \\ \rightarrow & \text{disjoint } (\text{add } (32, a, \text{asl } (32, x, cnt)), \text{opsz}, b, l) \end{aligned}$$

THEOREM: disjoint-2~asl

$$\begin{aligned} & (\text{disjoint } (a, m, b, n) \\ & \wedge \quad (\text{opsz} = \exp(2, cnt)) \\ & \wedge \quad (\text{nat-to-int } (x, 32) < (m \div \text{opsz})) \\ & \wedge \quad (\text{nat-to-int } (x, 32) \in \mathbf{N}) \\ & \wedge \quad (l \leq n)) \\ \rightarrow & \text{disjoint } (b, l, \text{add } (32, a, \text{asl } (32, x, cnt)), \text{opsz}) \end{aligned}$$

THEOREM: disjoint-3-asl

$$\begin{aligned} & (\text{disjoint } (a, m, b, n) \\ & \wedge \quad (\text{opsz} = \exp(2, cnt)) \\ & \wedge \quad (\text{nat-to-int } (x, 32) < (m \div \text{opsz})) \\ & \wedge \quad (\text{nat-to-int } (x, 32) \in \mathbf{N}) \\ & \wedge \quad ((k + l) \leq n)) \\ \rightarrow & \text{disjoint } (\text{add } (32, a, \text{asl } (32, x, cnt)), \text{opsz}, \text{add } (32, b, k), l) \end{aligned}$$

THEOREM: disjoint-3~asl

$$\begin{aligned} & (\text{disjoint } (a, m, b, n) \\ & \wedge \quad (\text{opsz} = \exp(2, cnt)) \\ & \wedge \quad (\text{nat-to-int } (x, 32) < (m \div \text{opsz})) \\ & \wedge \quad (\text{nat-to-int } (x, 32) \in \mathbf{N}) \\ & \wedge \quad ((k + l) \leq n)) \\ \rightarrow & \text{disjoint } (\text{add } (32, b, k), l, \text{add } (32, a, \text{asl } (32, x, cnt)), \text{opsz}) \end{aligned}$$

THEOREM: disjoint-9-asl

$$\begin{aligned} & (\text{disjoint } (a, m, \text{add } (32, b, k), n) \\ & \wedge \quad (\text{opsz} = \exp(2, cnt)) \\ & \wedge \quad (\text{nat-to-int } (x, 32) < (m \div \text{opsz})) \\ & \wedge \quad (\text{nat-to-int } (x, 32) \in \mathbf{N}) \\ & \wedge \quad ((l + \text{index-n } (0, k)) \leq n)) \\ \rightarrow & \text{disjoint } (\text{add } (32, a, \text{asl } (32, x, cnt)), \text{opsz}, b, l) \end{aligned}$$

THEOREM: disjoint-9~asl

$$\begin{aligned}
& (\text{disjoint}(a, m, \text{add}(32, b, k), n) \\
& \quad \wedge \quad (\text{opsz} = \exp(2, \text{cnt})) \\
& \quad \wedge \quad (\text{nat-to-int}(x, 32) < (m \div \text{opsz})) \\
& \quad \wedge \quad (\text{nat-to-int}(x, 32) \in \mathbf{N}) \\
& \quad \wedge \quad ((l + \text{index-n}(0, k)) \leq n)) \\
& \rightarrow \quad \text{disjoint}(b, l, \text{add}(32, a, \text{asl}(32, x, \text{cnt})), \text{opsz})
\end{aligned}$$

THEOREM: disjoint-11-asl

$$\begin{aligned}
& (\text{disjoint}(a, m, \text{add}(32, b, k), n) \\
& \quad \wedge \quad (\text{opsz} = \exp(2, \text{cnt})) \\
& \quad \wedge \quad (\text{nat-to-int}(x, 32) < (m \div \text{opsz})) \\
& \quad \wedge \quad (\text{nat-to-int}(x, 32) \in \mathbf{N}) \\
& \quad \wedge \quad ((l + \text{index-n}(k1, k)) \leq n)) \\
& \rightarrow \quad \text{disjoint}(\text{add}(32, a, \text{asl}(32, x, \text{cnt})), \text{opsz}, \text{add}(32, b, k1), l)
\end{aligned}$$

THEOREM: disjoint-11~asl

$$\begin{aligned}
& (\text{disjoint}(a, m, \text{add}(32, b, k), n) \\
& \quad \wedge \quad (\text{opsz} = \exp(2, \text{cnt})) \\
& \quad \wedge \quad (\text{nat-to-int}(x, 32) < (m \div \text{opsz})) \\
& \quad \wedge \quad (\text{nat-to-int}(x, 32) \in \mathbf{N}) \\
& \quad \wedge \quad ((l + \text{index-n}(k1, k)) \leq n)) \\
& \rightarrow \quad \text{disjoint}(\text{add}(32, b, k1), l, \text{add}(32, a, \text{asl}(32, x, \text{cnt})), \text{opsz})
\end{aligned}$$

EVENT: Disable times-plus-lessp-cancel.

; a set of rewrite rules for disjoint0 and disjoint.

THEOREM: disjoint0-x-x

$$\text{disjoint0}(x, m, x) = (m \simeq 0)$$

THEOREM: disjoint0-deduction0

$$\text{disjoint0}(x, 0, y)$$

THEOREM: disjoint0-deduction1

$$\begin{aligned}
& (\text{disjoint0}(x, m, \text{add}(32, x, y)) = \text{disjoint0}(0, m, y)) \\
& \wedge \quad (\text{disjoint0}(\text{add}(32, x, y), m, x) = \text{disjoint0}(y, m, 0))
\end{aligned}$$

THEOREM: disjoint0-deduction2

$$\text{disjoint0}(\text{add}(32, x, y), m, \text{add}(32, x, z)) = \text{disjoint0}(y, m, z)$$

THEOREM: disjoint-x-x

$$\text{disjoint}(x, m, x, n) = ((m \simeq 0) \vee (n \simeq 0))$$

THEOREM: disjoint-deduction0

$$\text{disjoint}(x, m, y, 0) \wedge \text{disjoint}(x, 0, y, n)$$

THEOREM: disjoint-deduction1

$(\text{disjoint}(x, m, \text{add}(32, x, y), n) = \text{disjoint}(0, m, y, n))$
 $\wedge (\text{disjoint}(\text{add}(32, x, y), m, x, n) = \text{disjoint}(y, m, 0, n))$

THEOREM: disjoint-deduction2

$\text{disjoint}(\text{add}(32, x, y), m, \text{add}(32, x, z), n) = \text{disjoint}(y, m, z, n)$

;

INDEX-N

THEOREM: index-n-0

$(\text{index-n}(x, 0) = \text{head}(x, 32)) \wedge (\text{index-n}(0, x) = \text{neg}(32, x))$

THEOREM: index-n-x-x

$\text{index-n}(x, x) = 0$

THEOREM: index-n-deduction0

$(\text{index-n}(x, \text{neg}(32, y)) = \text{index-n}(\text{add}(32, x, y), 0))$
 $\wedge (\text{index-n}(\text{neg}(32, x), y) = \text{index-n}(0, \text{add}(32, x, y)))$

THEOREM: index-n-deduction1

$(\text{index-n}(\text{add}(32, x, y), x) = \text{index-n}(y, 0))$
 $\wedge (\text{index-n}(y, \text{add}(32, y, x)) = \text{index-n}(0, x))$

THEOREM: index-n-deduction2

$\text{index-n}(\text{add}(32, z, x), \text{add}(32, z, y)) = \text{index-n}(x, y)$

THEOREM: disjoint-deduction3

$\text{disjoint}(\text{add}(32, y, x), m, \text{add}(32, z, x), n) = \text{disjoint}(y, m, z, n)$

EVENT: Disable index-n.

;

READ-MEM/WRITE-MEM WITH MEM-LST/MEM-ILST

;

; starting at address a, the contents of the memory are equal to the elements
; of lst.

DEFINITION:

$\text{mem-lst}(opsz, a, mem, n, lst)$

= **if** $n \simeq 0$ **then** $lst \simeq \text{nil}$

else $(\text{read-mem}(a, mem, opsz) = \text{car}(lst))$

$\wedge \text{mem-lst}(opsz, \text{add}(L, a, opsz), mem, n - 1, \text{cdr}(lst))$ **endif**

DEFINITION:

$\text{mem-ilst}(opsz, a, mem, n, lst)$

= **if** $n \simeq 0$ **then** $lst \simeq \text{nil}$

else $(\text{nat-to-int}(\text{read-mem}(a, mem, opsz), 8 * opsz) = \text{car}(lst))$

$\wedge \text{mem-ilst}(opsz, \text{add}(L, a, opsz), mem, n - 1, \text{cdr}(lst))$ **endif**

; every element in lst is in "good" range.

THEOREM: mem-1st-nat-range
 $(\text{mem-1st}(\text{opsz}, a, \text{mem}, n, \text{lst}) \wedge ((8 * \text{opsz}) \leq k))$
 $\rightarrow \text{nat-range}(\text{get-nth}(i, \text{lst}), k)$

THEOREM: mem-ilst-int-range
 $(\text{mem-ilst}(\text{opsz}, a, \text{mem}, n, \text{lst}) \wedge (\text{oplen} = (8 * \text{opsz})))$
 $\rightarrow \text{int-range}(\text{get-nth}(i, \text{lst}), \text{oplen})$

; every element in lst is a natural number.

THEOREM: mem-1st-numberp
 $\text{mem-1st}(\text{opsz}, a, \text{mem}, n, \text{lst}) \rightarrow (\text{get-nth}(i, \text{lst}) \in \mathbf{N})$

; every element in lst is bounded by $2^{(8*\text{opsz})}$.

THEOREM: mem-1st-lessp
 $(\text{mem-1st}(\text{opsz}, a, \text{mem}, n, \text{lst}) \wedge (k = \exp(2, 8 * \text{opsz})))$
 $\rightarrow (\text{get-nth}(i, \text{lst}) < k)$

; every element in lst is an integer.

THEOREM: mem-ilst-integerp
 $\text{mem-ilst}(\text{opsz}, a, \text{mem}, n, \text{lst}) \rightarrow \text{integerp}(\text{get-nth}(i, \text{lst}))$

; a trivial, but useful, instantiation!

THEOREM: readm-mem-1st
 $\text{mem-1st}(\text{opsz}, a, \text{mem}, n, \text{readm-mem}(\text{opsz}, a, \text{mem}, n))$

; head with mem-1st/mem-ilst.

THEOREM: head-mem-1st
 $\text{mem-1st}(\text{opsz}, \text{head}(x, 32), \text{mem}, n, \text{lst}) = \text{mem-1st}(\text{opsz}, x, \text{mem}, n, \text{lst})$

THEOREM: head-mem-ilst
 $\text{mem-ilst}(\text{opsz}, \text{head}(x, 32), \text{mem}, n, \text{lst}) = \text{mem-ilst}(\text{opsz}, x, \text{mem}, n, \text{lst})$

THEOREM: read-mem-non-numberp
 $(x \notin \mathbf{N}) \rightarrow (\text{read-mem}(x, \text{mem}, k) = \text{read-mem}(0, \text{mem}, k))$

THEOREM: mem-1st-non-numberp
 $(x \notin \mathbf{N}) \rightarrow (\text{mem-1st}(\text{opsz}, x, \text{mem}, n, \text{lst}) = \text{mem-1st}(\text{opsz}, 0, \text{mem}, n, \text{lst}))$

THEOREM: mem-1st-same
 $(\text{mem-1st}(\text{opsz}, x, \text{mem}, n, \text{lst}) \wedge (\text{nat-to-uint}(x) = \text{nat-to-uint}(y)))$
 $\rightarrow \text{mem-1st}(\text{opsz}, y, \text{mem}, n, \text{lst})$

EVENT: Disable read-mem-non-numberp.

EVENT: Disable mem-lst-non-numberp.

EVENT: Disable mem-lst-same.

```
;                                CANONICAL FORMS
; Sometimes, it is possible to have more than one representation
; for one concept. We have to tell the prover only to use the canonical
; representations.
```

THEOREM: pc-byte-read=pc-read-mem-1
pc-byte-read(*addr*, *mem*) = pc-read-mem(*addr*, *mem*, 1)

THEOREM: byte-read=read-mem-1
byte-read(*addr*, *mem*) = read-mem(*addr*, *mem*, 1)

THEOREM: byte-write=write-mem-1
byte-write(*value*, *addr*, *mem*) = write-mem(*value*, *addr*, *mem*, 1)

THEOREM: byte-writep=write-memp-1
byte-writep(*x*, *mem*) = write-memp(*x*, *mem*, 1)

```
;      GET-LST, PUT-LST, GET-VALS, PUT-VALS, MCAR, and MCDR
```

DEFINITION:
get-lst(*opsz*, *m*, *lst*, *n*)
= if *n* $\simeq$ 0 then 0
 else app(8 * *opsz*,
 get-nth(*m* + (*n* - 1), *lst*),
 get-lst(*opsz*, *m*, *lst*, *n* - 1)) endif

DEFINITION:
put-lst(*opsz*, *v*, *n*, *lst*, *k*)
= if *k* $\simeq$ 0 then *lst*
 else put-lst(*opsz*,
 tail(*v*, 8 * *opsz*),
 n,
 put-nth(head(*v*, 8 * *opsz*), *n* + (*k* - 1), *lst*),
 k - 1) endif

DEFINITION:
get-vals(*m*, *lst*, *n*)
= if *n* $\simeq$ 0 then nil
 else append(get-vals(*m*, *lst*, *n* - 1),
 list(get-nth(*m* + (*n* - 1), *lst*))) endif

DEFINITION:

```
put-vals(vals, m, lst, n)
=  if  $n \simeq 0$  then lst
   else put-vals(vals,
                 m,
                 put-nth(get-nth( $n - 1$ , vals),  $m + (n - 1)$ , lst),
                  $n - 1$ ) endif
```

DEFINITION:

```
bv-to-lst(opsz, bv, n)
=  if  $n \simeq 0$  then nil
   else append(bv-to-lst(opsz, tail(bv,  $8 * opsz$ ),  $n - 1$ ),
               list(head(bv,  $8 * opsz$ ))) endif
```

DEFINITION:

```
lst-to-bv(opsz, lst, n)
=  if  $n \simeq 0$  then 0
   else app( $8 * opsz$ ,
            get-nth( $n - 1$ , lst),
            lst-to-bv(opsz, lst,  $n - 1$ )) endif
```

; mcar returns the list consisting of the first *n* elements of *lst*.

DEFINITION:

```
mcar(n, lst)
=  if  $n \simeq 0$  then nil
   else cons(car(lst), mcar( $n - 1$ , cdr(lst))) endif
```

; mcdr returns the list discarding the first *n* elements of *lst*.

DEFINITION:

```
mcdr(n, lst)
=  if  $n \simeq 0$  then lst
   else mcdr( $n - 1$ , cdr(lst)) endif
```

; a predicate to recognize proper list.

DEFINITION:

```
proper-lstp(lst)
=  if lst  $\simeq$  nil then lst = nil
   else proper-lstp(cdr(lst)) endif
```

THEOREM: append-len

$\text{len}(\text{append}(x, y)) = (\text{len}(x) + \text{len}(y))$

THEOREM: get-vals-len

$\text{len}(\text{get-vals}(m, \text{lst}, n)) = \text{fix}(n)$

THEOREM: bv-to-lst-len

$$\text{len}(\text{bv-to-lst}(\text{opsz}, \text{bv}, n)) = \text{fix}(n)$$

THEOREM: get-vals-proper-lstp

$$\text{proper-lstp}(\text{get-vals}(m, \text{lst}, n))$$

THEOREM: bv-to-lst-proper-lstp

$$\text{proper-lstp}(\text{bv-to-lst}(\text{opsz}, \text{bv}, n))$$

THEOREM: mcdl-listp-len

$$\text{listp}(\text{mcdl}(n, \text{lst})) = (n < \text{len}(\text{lst}))$$

THEOREM: cdr-mcdl

$$\text{cdr}(\text{mcdl}(n, \text{lst})) = \text{mcdl}(1 + n, \text{lst})$$

THEOREM: mcar-mcar

$$(m \leq n) \rightarrow (\text{mcar}(m, \text{mcar}(n, \text{lst})) = \text{mcar}(m, \text{lst}))$$

THEOREM: mcdl-mcdl

$$\text{mcdl}(n, \text{mcdl}(m, \text{lst})) = \text{mcdl}(m + n, \text{lst})$$

THEOREM: mcar-nth

$$\text{get-nth}(n, \text{mcar}(m, \text{lst}))$$

$$= \text{if } n < m \text{ then } \text{get-nth}(n, \text{lst}) \\ \text{else } 0 \text{ endif}$$

THEOREM: mcdl-nth

$$\text{get-nth}(n, \text{mcdl}(m, \text{lst})) = \text{get-nth}(m + n, \text{lst})$$

THEOREM: get-lst-cdr

$$\text{get-lst}(\text{opsz}, m, \text{cdr}(\text{lst}), n) = \text{get-lst}(\text{opsz}, 1 + m, \text{lst}, n)$$

THEOREM: get-lst-mcdl

$$\text{get-lst}(\text{opsz}, m, \text{mcdl}(j, \text{lst}), n) = \text{get-lst}(\text{opsz}, m + j, \text{lst}, n)$$

THEOREM: get-lst-mcar

$$((m + n) \leq j)$$

$$\rightarrow (\text{get-lst}(\text{opsz}, m, \text{mcar}(j, \text{lst}), n) = \text{get-lst}(\text{opsz}, m, \text{lst}, n))$$

THEOREM: get-vals-cdr

$$\text{get-vals}(m, \text{cdr}(\text{lst}), n) = \text{get-vals}(1 + m, \text{lst}, n)$$

THEOREM: get-vals-mcdl

$$\text{get-vals}(m, \text{mcdl}(j, \text{lst}), n) = \text{get-vals}(m + j, \text{lst}, n)$$

THEOREM: get-vals-mcar

$$((m + n) \leq j) \rightarrow (\text{get-vals}(m, \text{mcar}(j, \text{lst}), n) = \text{get-vals}(m, \text{lst}, n))$$

THEOREM: get-nth-append
 $\text{get-nth}(i, \text{append}(x, y))$
 $= \text{if } i < \text{len}(x) \text{ then } \text{get-nth}(i, x)$
 $\quad \text{else } \text{get-nth}(i - \text{len}(x), y) \text{ endif}$

THEOREM: get-vals-append
 $(n \leq \text{len}(x)) \rightarrow (\text{get-vals}(0, \text{append}(x, y), n) = \text{get-vals}(0, x, n))$

THEOREM: put-vals-append
 $(n \leq \text{len}(x)) \rightarrow (\text{put-vals}(\text{append}(x, y), m, \text{lst}, n) = \text{put-vals}(x, m, \text{lst}, n))$

; another induction hint for many theorems about memory.

DEFINITION:
 $\text{mem-induct2}(\text{opsz}, \text{addr}, i, n, \text{lst}, j)$
 $= \text{if } n \simeq 0 \text{ then } t$
 $\quad \text{else } \text{mem-induct2}(\text{opsz},$
 $\quad \quad \text{add}(32, \text{addr}, \text{opsz}),$
 $\quad \quad i - 1,$
 $\quad \quad n - 1,$
 $\quad \quad \text{cdr}(\text{lst}),$
 $\quad \quad j - \text{opsz}) \text{ endif}$

THEOREM: read-mem-lst-la
 $(\text{mem-lst}(\text{opsz}, a, \text{mem}, n, \text{lst})$
 $\quad \wedge ((j \bmod \text{opsz}) = 0)$
 $\quad \wedge ((j \div \text{opsz}) < n))$
 $\rightarrow (\text{read-mem}(\text{add}(32, a, j), \text{mem}, \text{opsz}) = \text{get-nth}(j \div \text{opsz}, \text{lst}))$

THEOREM: mem-lst-get-lst0
 $(\text{mem-lst}(1, a, \text{mem}, n, \text{lst}) \wedge (k \leq n))$
 $\rightarrow (\text{nat-to-uint}(\text{read-mem}(a, \text{mem}, k)) = \text{get-lst}(1, 0, \text{lst}, k))$

THEOREM: mem-lst-get-lst
 $(\text{mem-lst}(1, a, \text{mem}, n, \text{lst}) \wedge ((\text{nat-to-uint}(j) + k) \leq n))$
 $\rightarrow (\text{nat-to-uint}(\text{read-mem}(\text{add}(32, a, j), \text{mem}, k))$
 $\quad = \text{get-lst}(1, \text{nat-to-uint}(j), \text{lst}, k))$

THEOREM: mem-lst-get-vals0
 $(\text{mem-lst}(1, a, \text{mem}, n, \text{lst}) \wedge (k \leq n))$
 $\rightarrow (\text{bv-to-lst}(1, \text{read-mem}(a, \text{mem}, k), k) = \text{get-vals}(0, \text{lst}, k))$

THEOREM: mem-lst-get-vals
 $(\text{mem-lst}(1, a, \text{mem}, n, \text{lst}) \wedge ((\text{nat-to-uint}(j) + k) \leq n))$
 $\rightarrow (\text{bv-to-lst}(1, \text{read-mem}(\text{add}(32, a, j), \text{mem}, k), k)$
 $\quad = \text{get-vals}(\text{nat-to-uint}(j), \text{lst}, k))$

; read is equivalent to get-nth.

THEOREM: read-mem-ilst0

$$\begin{aligned} & (\text{mem-ilst}(\text{opsz}, a, \text{mem}, n, \text{lst}) \wedge (n \neq 0)) \\ \rightarrow & (\text{nat-to-uint}(\text{read-mem}(a, \text{mem}, \text{opsz})) = \text{get-nth}(0, \text{lst})) \end{aligned}$$

THEOREM: iread-mem-get0

$$\begin{aligned} & (\text{mem-ilst}(\text{opsz}, a, \text{mem}, n, \text{lst}) \wedge (n \neq 0) \wedge (\text{oplen} = (8 * \text{opsz}))) \\ \rightarrow & (\text{nat-to-int}(\text{read-mem}(a, \text{mem}, \text{opsz}), \text{oplen}) = \text{get-nth}(0, \text{lst})) \end{aligned}$$

THEOREM: read-mem-ilst

$$\begin{aligned} & (\text{mem-ilst}(\text{opsz}, a, \text{mem}, n, \text{lst}) \\ & \wedge ((\text{nat-to-uint}(j) \bmod \text{opsz}) = 0) \\ & \wedge ((\text{nat-to-uint}(j) \div \text{opsz}) < n)) \\ \rightarrow & (\text{nat-to-uint}(\text{read-mem}(\text{add}(32, a, j), \text{mem}, \text{opsz})) \\ & = \text{get-nth}(\text{nat-to-uint}(j) \div \text{opsz}, \text{lst})) \end{aligned}$$

THEOREM: read-mem-ilst-int

$$\begin{aligned} & (\text{mem-ilst}(\text{opsz}, a, \text{mem}, n, \text{lst}) \\ & \wedge ((\text{nat-to-int}(j, 32) \bmod \text{opsz}) = 0) \\ & \wedge ((\text{nat-to-int}(j, 32) \div \text{opsz}) < n) \\ & \wedge (\text{nat-to-int}(j, 32) \in \mathbf{N})) \\ \rightarrow & (\text{nat-to-uint}(\text{read-mem}(\text{add}(32, a, j), \text{mem}, \text{opsz})) \\ & = \text{get-nth}(\text{nat-to-int}(j, 32) \div \text{opsz}, \text{lst})) \end{aligned}$$

THEOREM: read-mem-ilst

$$\begin{aligned} & (\text{mem-ilst}(\text{opsz}, a, \text{mem}, n, \text{lst}) \\ & \wedge (\text{oplen} = (8 * \text{opsz})) \\ & \wedge ((\text{nat-to-uint}(j) \bmod \text{opsz}) = 0) \\ & \wedge ((\text{nat-to-uint}(j) \div \text{opsz}) < n)) \\ \rightarrow & (\text{nat-to-int}(\text{read-mem}(\text{add}(32, a, j), \text{mem}, \text{opsz}), \text{oplen}) \\ & = \text{get-nth}(\text{nat-to-uint}(j) \div \text{opsz}, \text{lst})) \end{aligned}$$

THEOREM: read-mem-ilst-int

$$\begin{aligned} & (\text{mem-ilst}(\text{opsz}, a, \text{mem}, n, \text{lst}) \\ & \wedge (\text{oplen} = (8 * \text{opsz})) \\ & \wedge ((\text{nat-to-uint}(j) \bmod \text{opsz}) = 0) \\ & \wedge ((\text{nat-to-uint}(j) \div \text{opsz}) < n) \\ & \wedge (\text{nat-to-int}(j, 32) \in \mathbf{N})) \\ \rightarrow & (\text{nat-to-int}(\text{read-mem}(\text{add}(32, a, j), \text{mem}, \text{opsz}), \text{oplen}) \\ & = \text{get-nth}(\text{nat-to-uint}(j) \div \text{opsz}, \text{lst})) \end{aligned}$$

; write to some location else does not affect mem-ilst.

THEOREM: write-else-mem-ilst

$$\begin{aligned} & \text{disjoint}(a, \text{opsz} * n, x, m) \\ \rightarrow & (\text{mem-ilst}(\text{opsz}, a, \text{write-mem}(\text{value}, x, \text{mem}, m), n, \text{lst}) \\ & = \text{mem-ilst}(\text{opsz}, a, \text{mem}, n, \text{lst})) \end{aligned}$$

THEOREM: write-else-mem-ilst

$$\begin{aligned} & \text{disjoint}(a, \text{opsz} * n, x, m) \\ \rightarrow & (\text{mem-ilst}(\text{opsz}, a, \text{write-mem}(\text{value}, x, \text{mem}, m), n, \text{lst}) \\ & = \text{mem-ilst}(\text{opsz}, a, \text{mem}, n, \text{lst})) \end{aligned}$$

; some conditions that makes disjoint true.

THEOREM: disjoint0-leq

$$(((a + m) \leq 4294967296) \wedge (b < a)) \rightarrow \text{disjoint0}(a, m, b)$$

THEOREM: disjoint-leq

$$(((a + m) \leq 4294967296) \wedge ((b + n) \leq a)) \rightarrow \text{disjoint}(a, m, b, n)$$

THEOREM: disjoint-leq1

$$(((a + m) \leq 4294967296) \wedge ((b + n) \leq a)) \rightarrow \text{disjoint}(b, n, a, m)$$

THEOREM: plus-times-lessp

$$\begin{aligned} & (((x + (y * z)) < w) \wedge (z1 < z)) \\ \rightarrow & ((x + y + (y * z1)) < w) \end{aligned}$$

THEOREM: write-mem-lst-la

$$\begin{aligned} & (\text{mem-lst}(\text{opsz}, a, \text{mem}, n, \text{lst}) \\ & \wedge \text{uint-range}(\text{opsz} * n, 32) \\ & \wedge (v1 = \text{head}(v, 8 * \text{opsz})) \\ & \wedge (j = (\text{opsz} * i)) \\ & \wedge (i < n) \\ & \wedge (v \in \mathbf{N})) \\ \rightarrow & \text{mem-lst}(\text{opsz}, a, \text{write-mem}(v, \text{add}(32, a, j), \text{mem}, \text{opsz}), n, \text{put-nth}(v1, i, \text{lst})) \end{aligned}$$

DEFINITION:

mem-induct3($v, v1, a, \text{mem}, i, \text{lst}, k$)

$$\begin{aligned} = & \text{if } k \simeq 0 \text{ then t} \\ & \text{else mem-induct3}(\text{tail}(v, 8), \\ & \quad \text{tail}(v1, 8), \\ & \quad a, \\ & \quad \text{byte-write}(v, \text{add}(32, a, i + (k - 1)), \text{mem}), \\ & \quad i, \\ & \quad \text{put-nth}(\text{head}(v, 8), i + (k - 1), \text{lst}), \\ & \quad k - 1) \text{ endif} \end{aligned}$$

THEOREM: mem-lst-put-lst

$$\begin{aligned} & (\text{mem-lst}(1, a, \text{mem}, n, \text{lst}) \\ & \wedge \text{uint-range}(n, 32) \\ & \wedge \text{nat-range}(v, 8 * k) \\ & \wedge (\text{nat-to-uint}(j) = i) \end{aligned}$$

$$\begin{aligned}
& \wedge (v1 = \text{nat-to-uint}(v)) \\
& \wedge ((i + k) \leq n) \\
& \wedge (j \in \mathbf{N}) \\
& \wedge (v \in \mathbf{N}) \\
\rightarrow & \text{mem-lst}(1, a, \text{write-mem}(v, \text{add}(32, a, j), \text{mem}, k), n, \text{put-lst}(1, v1, i, \text{lst}, k))
\end{aligned}$$

DEFINITION:

$$\begin{aligned}
& \text{mem-induct4}(v, \text{vals}, a, \text{mem}, i, \text{lst}, k) \\
= & \text{if } k \simeq 0 \text{ then t} \\
& \text{else mem-induct4}(\text{tail}(v, 8), \\
& \quad \text{get-vals}(0, \text{vals}, k - 1), \\
& \quad a, \\
& \quad \text{byte-write}(v, \text{add}(32, a, i + (k - 1)), \text{mem}), \\
& \quad i, \\
& \quad \text{put-nth}(\text{head}(v, 8), i + (k - 1), \text{lst}), \\
& \quad k - 1) \text{ endif}
\end{aligned}$$

THEOREM: get-vals-0

$$((\text{len}(\text{lst}) = n) \wedge \text{proper-lstp}(\text{lst})) \rightarrow (\text{get-vals}(0, \text{lst}, n) = \text{lst})$$

THEOREM: mem-lst-put-vals

$$\begin{aligned}
& (\text{mem-lst}(1, a, \text{mem}, n, \text{lst}) \\
& \wedge \text{uint-range}(n, 32) \\
& \wedge \text{nat-range}(v, 8 * k) \\
& \wedge (\text{nat-to-uint}(j) = i) \\
& \wedge (\text{vals} = \text{bv-to-lst}(1, v, k)) \\
& \wedge ((i + k) \leq n) \\
& \wedge (j \in \mathbf{N}) \\
& \wedge (v \in \mathbf{N})) \\
\rightarrow & \text{mem-lst}(1, a, \text{write-mem}(v, \text{add}(32, a, j), \text{mem}, k), n, \text{put-vals}(\text{vals}, i, \text{lst}, k))
\end{aligned}$$

EVENT: Disable get-vals-0.

THEOREM: write-mem-lst

$$\begin{aligned}
& (\text{mem-lst}(\text{opsz}, a, \text{mem}, n, \text{lst}) \\
& \wedge \text{uint-range}(\text{opsz} * n, 32) \\
& \wedge \text{nat-range}(v, 8 * \text{opsz}) \\
& \wedge (\text{nat-to-uint}(j) = (\text{opsz} * i)) \\
& \wedge (v1 = \text{nat-to-uint}(v)) \\
& \wedge (i < n) \\
& \wedge (j \in \mathbf{N}) \\
& \wedge (v \in \mathbf{N})) \\
\rightarrow & \text{mem-lst}(\text{opsz}, a, \text{write-mem}(v, \text{add}(32, a, j), \text{mem}, \text{opsz}), n, \text{put-nth}(v1, i, \text{lst}))
\end{aligned}$$

THEOREM: write-mem-ilst

$$\begin{aligned}
& (\text{mem-ilst } (opsz, a, mem, n, lst) \\
& \quad \wedge \text{ uint-range } (opsz * n, 32) \\
& \quad \wedge (\text{nat-to-uint } (j) = (opsz * i)) \\
& \quad \wedge \text{ nat-range } (v, 8 * opsz) \\
& \quad \wedge (v1 = \text{nat-to-int } (v, 8 * opsz)) \\
& \quad \wedge (i < n) \\
& \quad \wedge (j \in \mathbf{N}) \\
& \quad \wedge (v \in \mathbf{N})) \\
& \rightarrow \text{ mem-ilst } (opsz, \\
& \quad a, \\
& \quad \text{write-mem } (v, \text{add } (32, a, j), mem, opsz), \\
& \quad n, \\
& \quad \text{put-nth } (v1, i, lst))
\end{aligned}$$

THEOREM: write-mem-lst0

$$\begin{aligned}
& (\text{mem-lst } (opsz, a, mem, n, lst) \\
& \quad \wedge \text{ uint-range } (opsz * n, 32) \\
& \quad \wedge \text{ nat-range } (v, 8 * opsz) \\
& \quad \wedge (v1 = \text{nat-to-uint } (v)) \\
& \quad \wedge (n \neq 0) \\
& \quad \wedge (v \in \mathbf{N})) \\
& \rightarrow \text{ mem-lst } (opsz, a, \text{write-mem } (v, a, mem, opsz), n, \text{put-nth } (v1, 0, lst))
\end{aligned}$$

THEOREM: write-mem-ilst0

$$\begin{aligned}
& (\text{mem-ilst } (opsz, a, mem, n, lst) \\
& \quad \wedge \text{ uint-range } (opsz * n, 32) \\
& \quad \wedge \text{ nat-range } (v, 8 * opsz) \\
& \quad \wedge (v1 = \text{nat-to-int } (v, 8 * opsz)) \\
& \quad \wedge (n \neq 0) \\
& \quad \wedge (v \in \mathbf{N})) \\
& \rightarrow \text{ mem-ilst } (opsz, a, \text{write-mem } (v, a, mem, opsz), n, \text{put-nth } (v1, 0, lst))
\end{aligned}$$

THEOREM: write-mem-lst-int

$$\begin{aligned}
& (\text{mem-lst } (opsz, a, mem, n, lst) \\
& \quad \wedge \text{ uint-range } (opsz * n, 32) \\
& \quad \wedge ((opsz * i) = \text{nat-to-int } (j, 32)) \\
& \quad \wedge (\text{nat-to-int } (j, 32) \in \mathbf{N}) \\
& \quad \wedge \text{ nat-range } (v, 8 * opsz) \\
& \quad \wedge (v1 = \text{nat-to-uint } (v)) \\
& \quad \wedge (i < n) \\
& \quad \wedge (opsz \neq 0) \\
& \quad \wedge (v \in \mathbf{N})) \\
& \rightarrow \text{ mem-lst } (opsz, a, \text{write-mem } (v, \text{add } (32, a, j), mem, opsz), n, \text{put-nth } (v1, i, lst))
\end{aligned}$$

THEOREM: write-mem-ilst-int

```

(mem-ilst (opsz, a, mem, n, lst)
  ∧ uint-range (opsz * n, 32)
  ∧ ((opsz * i) = nat-to-int (j, 32))
  ∧ (nat-to-int (j, 32) ∈ ℕ)
  ∧ nat-range (v, 8 * opsz)
  ∧ (v1 = nat-to-int (v, 8 * opsz))
  ∧ (i < n)
  ∧ (opsz ≠ 0)
  ∧ (v ∈ ℕ))
→ mem-ilst (opsz,
             a,
             write-mem (v, add (32, a, j), mem, opsz),
             n,
             put-nth (v1, i, lst))

;                               INTEGER VIEWS
;                               A + OPSZ * I
; to deal with the address calculation a+2*i or a+4*i, we introduce
; the following set of lemmas.

```

THEOREM: read-mem-lst-asl

```

(mem-lst (opsz, a, mem, n, lst)
  ∧ (opsz = exp (2, cnt))
  ∧ (nat-to-int (i, 32) < n)
  ∧ (nat-to-int (i, 32) ∈ ℕ))
→ (nat-to-uint (read-mem (add (32, a, asl (32, i, cnt)), mem, opsz))
   = get-nth (nat-to-int (i, 32), lst))

```

THEOREM: read-mem-ilst-asl

```

(mem-ilst (opsz, a, mem, n, lst)
  ∧ (opsz = exp (2, cnt))
  ∧ (oplen = (8 * opsz))
  ∧ (nat-to-int (i, 32) < n)
  ∧ (nat-to-int (i, 32) ∈ ℕ))
→ (nat-to-int (read-mem (add (32, a, asl (32, i, cnt)), mem, opsz), oplen)
   = get-nth (nat-to-int (i, 32), lst))

```

THEOREM: write-mem-lst-asl

```

(mem-lst (opsz, a, mem, n, lst)
  ∧ uint-range (opsz * n, 32)
  ∧ (opsz = exp (2, cnt))
  ∧ nat-range (v, 8 * opsz)
  ∧ (j = nat-to-int (i, 32))
  ∧ (v1 = nat-to-uint (v))

```

$$\begin{aligned}
& \wedge (j < n) \\
& \wedge (j \in \mathbf{N}) \\
& \wedge (v \in \mathbf{N}) \\
\rightarrow & \text{mem-ilst}(\text{opsz}, \\
& a, \\
& \text{write-mem}(v, \text{add}(32, a, \text{asl}(32, i, \text{cnt})), \text{mem}, \text{opsz}), \\
& n, \\
& \text{put-nth}(v1, j, \text{lst}))
\end{aligned}$$

THEOREM: write-mem-ilst-asl

$$\begin{aligned}
& (\text{mem-ilst}(\text{opsz}, a, \text{mem}, n, \text{lst}) \\
& \wedge \text{uint-range}(\text{opsz} * n, 32) \\
& \wedge (\text{opsz} = \exp(2, \text{cnt})) \\
& \wedge \text{nat-range}(v, 8 * \text{opsz}) \\
& \wedge (v1 = \text{nat-to-int}(v, 8 * \text{opsz})) \\
& \wedge (j = \text{nat-to-int}(i, 32)) \\
& \wedge (j < n) \\
& \wedge (v \in \mathbf{N}) \\
& \wedge (j \in \mathbf{N})) \\
\rightarrow & \text{mem-ilst}(\text{opsz}, \\
& a, \\
& \text{write-mem}(v, \text{add}(32, a, \text{asl}(32, i, \text{cnt})), \text{mem}, \text{opsz}), \\
& n, \\
& \text{put-nth}(v1, j, \text{lst}))
\end{aligned}$$

THEOREM: times-lessp-cancel1

$$((y * x) < (x + (z * x))) = ((x \neq 0) \wedge (y \leq z))$$

THEOREM: read-memp-ram1-asl

$$\begin{aligned}
& (\text{ram-addrp}(\text{addr}, \text{mem}, n) \\
& \wedge (\text{opsz} = \exp(2, \text{cnt})) \\
& \wedge (\text{nat-to-int}(x, 32) < (n \div \text{opsz})) \\
& \wedge (\text{nat-to-int}(x, 32) \in \mathbf{N})) \\
\rightarrow & \text{read-memp}(\text{add}(32, \text{addr}, \text{asl}(32, x, \text{cnt})), \text{mem}, \text{opsz})
\end{aligned}$$

THEOREM: read-memp-rom1-asl

$$\begin{aligned}
& (\text{rom-addrp}(\text{addr}, \text{mem}, n) \\
& \wedge (\text{opsz} = \exp(2, \text{cnt})) \\
& \wedge (\text{nat-to-int}(x, 32) < (n \div \text{opsz})) \\
& \wedge (\text{nat-to-int}(x, 32) \in \mathbf{N})) \\
\rightarrow & \text{read-memp}(\text{add}(32, \text{addr}, \text{asl}(32, x, \text{cnt})), \text{mem}, \text{opsz})
\end{aligned}$$

THEOREM: write-memp-ram1-asl

$$\begin{aligned}
& (\text{ram-addrp}(\text{addr}, \text{mem}, n) \\
& \wedge (\text{opsz} = \exp(2, \text{cnt}))
\end{aligned}$$

$\wedge$ (nat-to-int $(x, 32) < (n \div \text{opsz})$)
 $\wedge$ (nat-to-int $(x, 32) \in \mathbf{N}$)
 $\rightarrow$ write-memp (add (32, addr, asl (32, x, cnt)), mem, opsz)

; BASIC READM-MEM/WRITE-MEM EVENTS
 ;

THEOREM: writem-mem-maintain-pc-byte-readp
 pc-byte-readp $(x, \text{writem-mem}(\text{opsz}, \text{vlst}, \text{addr}, \text{mem})) = \text{pc-byte-readp}(x, \text{mem})$

THEOREM: writem-mem-maintain-pc-read-memp
 pc-read-memp $(x, \text{writem-mem}(\text{opsz}, \text{vlst}, \text{addr}, \text{mem}), n) = \text{pc-read-memp}(x, \text{mem}, n)$

THEOREM: writem-mem-maintain-byte-readp
 byte-readp $(x, \text{writem-mem}(\text{opsz}, \text{vlst}, \text{addr}, \text{mem})) = \text{byte-readp}(x, \text{mem})$

THEOREM: writem-mem-maintain-read-memp
 read-memp $(x, \text{writem-mem}(\text{opsz}, \text{vlst}, \text{addr}, \text{mem}), n) = \text{read-memp}(x, \text{mem}, n)$

THEOREM: writem-mem-maintain-byte-writep
 byte-writep $(x, \text{writem-mem}(\text{opsz}, \text{vlst}, \text{addr}, \text{mem})) = \text{byte-writep}(x, \text{mem})$

THEOREM: writem-mem-maintain-write-memp
 write-memp $(x, \text{writem-mem}(\text{opsz}, \text{vlst}, \text{addr}, \text{mem}), n) = \text{write-memp}(x, \text{mem}, n)$

THEOREM: writem-mem-maintain-rom-addrp
 rom-addrp $(x, \text{writem-mem}(\text{opsz}, \text{vlst}, \text{addr}, \text{mem}), n) = \text{rom-addrp}(x, \text{mem}, n)$

THEOREM: writem-mem-maintain-ram-addrp
 ram-addrp $(x, \text{writem-mem}(\text{opsz}, \text{vlst}, \text{addr}, \text{mem}), n) = \text{ram-addrp}(x, \text{mem}, n)$

THEOREM: pc-read-mem-writem-mem
 (write-memp (addr, mem, opsz * len (vlst)) $\wedge$ pc-read-memp (x, mem, n))
 $\rightarrow$ (pc-read-mem $(x, \text{writem-mem}(\text{opsz}, \text{vlst}, \text{addr}, \text{mem}), n)$
 $=$ pc-read-mem (x, mem, n))

THEOREM: writem-mem-mcode-addrp
 (pc-read-memp (pc, mem, len (lst)) $\wedge$ write-memp (addr, mem, opsz * len (vlst)))
 $\rightarrow$ (mcode-addrp (pc, writem-mem (opsz, vlst, addr, mem), lst)
 $=$ mcode-addrp (pc, mem, lst))

THEOREM: writem-else-mem-lst
 disjoint (addr, opsz1 * len (vlst), a, opsz * n)
 $\rightarrow$ (mem-lst (opsz, a, writem-mem (opsz1, vlst, addr, mem), n, lst)
 $=$ mem-lst (opsz, a, mem, n, lst))

THEOREM: writem-else-mem-ilst

disjoint($a, opsz * n, addr, opsz1 * \text{len}(vlst)$)
 $\rightarrow$ (mem-ilst($opsz, a, \text{writem-mem}(opsz1, vlst, addr, mem), n, lst$)
 $=$ mem-ilst($opsz, a, mem, n, lst$))

THEOREM: read-writem-mem

disjoint($addr, n, addr1, opsz * \text{len}(vlst)$)
 $\rightarrow$ (read-mem($addr, \text{writem-mem}(opsz, vlst, addr1, mem), n$)
 $=$ read-mem($addr, mem, n$))

THEOREM: readm-write-mem

disjoint($addr, opsz * n, addr1, k$)
 $\rightarrow$ (readm-mem($opsz, addr, \text{write-mem}(value, addr1, mem, k), n$)
 $=$ readm-mem($opsz, addr, mem, n$))

DEFINITION:

modn-lst(n, lst)
 $=$ **if** $lst \simeq \text{nil}$ **then nil**
else cons(head(car(lst), n), modn-lst($n, \text{cdr}(lst)$)) **endif**

THEOREM: modn-readm-rn

modn-lst($oplen, \text{readm-rn}(oplen, rnlst, rfile)$) = readm-rn($oplen, rnlst, rfile$)

THEOREM: readm-writem-mem

(uint-range($opsz * n, 32$) $\wedge$ ($n = \text{len}(vlst)$))
 $\rightarrow$ (readm-mem($opsz, addr, \text{writem-mem}(opsz, vlst, addr, mem), n$)
 $=$ modn-lst($8 * opsz, vlst$))

THEOREM: disjoint-leq-uint

((nat-to-uint(a) + m) ≤ 4294967296)
 $\wedge$ ((nat-to-uint(b) + n) $\leq$ nat-to-uint(a)))
 $\rightarrow$ disjoint(a, m, b, n)

THEOREM: disjoint-leq1-uint

((nat-to-uint(a) + m) ≤ 4294967296)
 $\wedge$ ((nat-to-uint(b) + n) $\leq$ nat-to-uint(a)))
 $\rightarrow$ disjoint(b, n, a, m)

EVENT: Disable disjoint-leq.

EVENT: Disable disjoint-leq1.

EVENT: Disable disjoint0-leq.

EVENT: Disable disjoint-commutativity.

EVENT: Disable disjoint-leq-uint.

EVENT: Disable disjoint-leq1-uint.

;
MEM-LST WITH MCAR AND MCDR

THEOREM: mem-lst-plus

$$\begin{aligned} & \text{mem-lst}(\text{opsz}, a, \text{mem}, m + n, \text{lst}) \\ &= (\text{mem-lst}(\text{opsz}, a, \text{mem}, m, \text{mcar}(m, \text{lst})) \\ & \quad \wedge \text{mem-lst}(\text{opsz}, \text{add}(32, a, \text{opsz} * m), \text{mem}, n, \text{mcdr}(m, \text{lst}))) \end{aligned}$$

THEOREM: mem-lst-mcar

$$\begin{aligned} & (\text{mem-lst}(\text{opsz}, a, \text{mem}, n, \text{lst}) \wedge (k \leq n)) \\ & \rightarrow \text{mem-lst}(\text{opsz}, a, \text{mem}, k, \text{mcar}(k, \text{lst})) \end{aligned}$$

THEOREM: mem-lst-mcar-1

$$\begin{aligned} & \text{mem-lst}(\text{opsz}, a, \text{mem}, n, \text{mcar}(n, \text{lst})) \\ & \rightarrow \text{mem-lst}(\text{opsz}, a, \text{mem}, n - 1, \text{mcar}(n - 1, \text{lst})) \end{aligned}$$

THEOREM: mem-lst-mcar-2

$$\begin{aligned} & (\text{mem-lst}(\text{opsz}, a, \text{mem}, n, \text{mcar}(n, \text{lst})) \wedge (k \leq n)) \\ & \rightarrow \text{mem-lst}(\text{opsz}, a, \text{mem}, n - k, \text{mcar}(n - k, \text{lst})) \end{aligned}$$

THEOREM: plus-difference

$$\begin{aligned} & (m + (n - m)) \\ &= \text{if } n < m \text{ then fix}(m) \\ & \quad \text{else fix}(n) \text{ endif} \end{aligned}$$

THEOREM: mem-lst-len

$$\text{mem-lst}(\text{opsz}, a, \text{mem}, n, \text{lst}) \rightarrow (n \not\leq \text{len}(\text{lst}))$$

THEOREM: mem-lst-mcdr

$$\begin{aligned} & (\text{mem-lst}(\text{opsz}, a, \text{mem}, n, \text{lst}) \wedge (i = (\text{opsz} * m))) \\ & \rightarrow \text{mem-lst}(\text{opsz}, \text{add}(32, a, i), \text{mem}, n - m, \text{mcdr}(m, \text{lst})) \end{aligned}$$

THEOREM: mem-lst-mcdr-0

$$\begin{aligned} & \text{mem-lst}(\text{opsz}, a, \text{mem}, n, \text{lst}) \\ & \rightarrow \text{mem-lst}(\text{opsz}, \text{add}(32, a, \text{opsz}), \text{mem}, n - 1, \text{mcdr}(1, \text{lst})) \end{aligned}$$

THEOREM: mem-lst-mcdr-uint

$$\begin{aligned} & (\text{mem-lst}(\text{opsz}, a, \text{mem}, n, \text{lst}) \\ & \quad \wedge (\text{nat-to-uint}(i) = (\text{opsz} * m)) \\ & \quad \wedge (\text{opsz} \neq 0)) \\ & \rightarrow \text{mem-lst}(\text{opsz}, \text{add}(32, a, i), \text{mem}, n - m, \text{mcdr}(m, \text{lst})) \end{aligned}$$

THEOREM: mem-lst-mcdr-1

$$\begin{aligned} & \text{mem-lst}(1, \text{add}(32, x, i), \text{mem}, n - j, \text{mcdr}(k, \text{lst})) \\ \rightarrow & \text{mem-lst}(1, \text{add}(32, x, \text{add}(32, i, 1)), \text{mem}, (n - 1) - j, \text{mcdr}(1 + k, \text{lst})) \end{aligned}$$

THEOREM: mem-lst-mcdr-uint-1

$$\begin{aligned} & (\text{mem-lst}(\text{opsz}, \text{add}(32, a, b), \text{mem}, n - j, \text{mcdr}(k, \text{lst})) \\ & \wedge (\text{nat-to-uint}(i) = (\text{opsz} * m)) \\ & \wedge (\text{opsz} \neq 0)) \\ \rightarrow & \text{mem-lst}(\text{opsz}, \\ & \quad \text{add}(32, a, \text{add}(32, b, i)), \\ & \quad \text{mem}, \\ & \quad n - (m + j), \\ & \quad \text{mcdr}(m + k, \text{lst})) \end{aligned}$$

EVENT: Disable mem-lst-len.

EVENT: Disable plus-difference.

; LSTCPY AND LSTMCPY

DEFINITION:

$$\begin{aligned} & \text{lstcpy}(i1, \text{lst1}, i2, \text{lst2}, n) \\ = & \text{if } n \simeq 0 \text{ then } \text{lst1} \\ & \text{else lstcpy}(1 + i1, \\ & \quad \text{put-nth}(\text{get-nth}(i2, \text{lst2}), i1, \text{lst1}), \\ & \quad 1 + i2, \\ & \quad \text{lst2}, \\ & \quad n - 1) \text{ endif} \end{aligned}$$

THEOREM: lstcpy-0

$$\text{lstcpy}(i1, \text{lst1}, i2, \text{lst2}, 0) = \text{lst1}$$

THEOREM: lstcpy-lstcpy

$$\begin{aligned} & ((j1 = (h1 + i1)) \wedge (j2 = (h1 + i2))) \\ \rightarrow & (\text{lstcpy}(j1, \text{lscpy}(i1, \text{lst1}, i2, \text{lst2}, h1), j2, \text{lst2}, h2) \\ & = \text{lscpy}(i1, \text{lst1}, i2, \text{lst2}, h1 + h2)) \end{aligned}$$

DEFINITION:

$$\begin{aligned} & \text{lstmcpy}(\text{opsz}, i1, \text{lst1}, i2, \text{lst2}, n) \\ = & \text{if } n \simeq 0 \text{ then } \text{lst1} \\ & \text{else lstmcpy}(\text{opsz}, \\ & \quad \text{opsz} + i1, \\ & \quad \text{lscpy}(i1, \text{lst1}, i2, \text{lst2}, \text{opsz}), \\ & \quad \text{opsz} + i2, \\ & \quad \text{lst2}, \\ & \quad n - 1) \text{ endif} \end{aligned}$$

THEOREM: lstmcpy-cpy

$$\text{lstmcpy}(h, i1, lst1, i2, lst2, n) = \text{lstcpy}(i1, lst1, i2, lst2, h * n)$$

THEOREM: put-commutativity

$$(j < i)$$

$$\rightarrow (\text{put-nth}(v1, i, \text{put-nth}(v2, j, lst)) = \text{put-nth}(v2, j, \text{put-nth}(v1, i, lst)))$$

THEOREM: lstcpy-put-nth

$$(i < j1)$$

$$\rightarrow (\text{put-nth}(v, i, \text{lstcpy}(j1, lst1, j2, lst2, n)) \\ = \text{lstcpy}(j1, \text{put-nth}(v, i, lst1), j2, lst2, n))$$

; a variant for lstcpy.

DEFINITION:

$$\begin{aligned} &\text{lstcpy1}(i1, lst1, i2, lst2, n) \\ &= \text{if } n \simeq 0 \text{ then } lst1 \\ &\quad \text{else lstcpy1}(i1, \\ &\quad \quad \text{put-nth}(\text{get-nth}(i2 + (n - 1), lst2), \\ &\quad \quad \quad i1 + (n - 1), \\ &\quad \quad \quad lst1), \\ &\quad \quad i2, \\ &\quad \quad lst2, \\ &\quad \quad n - 1) \text{ endif} \end{aligned}$$

THEOREM: lstcpy-add1

$$\begin{aligned} &\text{lstcpy}(i1, \text{put-nth}(\text{get-nth}(i2 + h, lst2), i1 + h, lst1), i2, lst2, h) \\ &= \text{lstcpy}(i1, lst1, i2, lst2, 1 + h) \end{aligned}$$

THEOREM: lstcpy-cpy1

$$\text{lstcpy}(i1, lst1, i2, lst2, n) = \text{lstcpy1}(i1, lst1, i2, lst2, n)$$

THEOREM: put-get-lst-is-cpy

$$\begin{aligned} &\text{mem-lst}(opsz, x, mem, n2, lst2) \\ &\rightarrow (\text{put-lst}(opsz, \text{get-lst}(opsz, i2, lst2, n), i1, lst1, n) \\ &\quad = \text{lstcpy}(i1, lst1, i2, lst2, n)) \end{aligned}$$

THEOREM: put-get-vals-is-cpy

$$\text{put-vals}(\text{get-vals}(i2, lst2, n), i1, lst1, n) = \text{lstcpy}(i1, lst1, i2, lst2, n)$$

EVENT: Disable lstcpy-cpy1.

; MM0V1-LST, MM0V-LST1, MM0VN-LST AND MM0VN-LST1

DEFINITION:

mmov1-*lst* (*i*, *lst1*, *lst2*, *n*)
 = **if** $n \simeq 0$ **then** *lst1*
 else mmov1-*lst* ($1 + i$, put-nth (get-nth (*i*, *lst2*), *i*, *lst1*), *lst2*, $n - 1$) **endif**
 ; the same as mov1-*lst*.

DEFINITION:

mmov1-*lst1* (*i*, *lst1*, *lst2*, *n*)
 = **if** $n \simeq 0$ **then** *lst1*
 else mmov1-*lst1* ($i - 1$,
 put-nth (get-nth ($i - 1$, *lst2*), $i - 1$, *lst1*),
 lst2,
 $n - 1$) **endif**

DEFINITION:

movn-*lst* (*n*, *lst1*, *lst2*, *i*) = put-vals (get-vals (*i*, *lst2*, *n*), *i*, *lst1*, *n*)

DEFINITION:

mmovn-*lst* (*h*, *lst1*, *lst2*, *i*, *nt*)
 = **if** $nt \simeq 0$ **then** *lst1*
 else mmovn-*lst* (*h*, movn-*lst* (*h*, *lst1*, *lst2*, *i*), *lst2*, $h + i$, $nt - 1$) **endif**

DEFINITION:

mmovn-*lst1* (*h*, *lst1*, *lst2*, *i*, *nt*)
 = **if** $nt \simeq 0$ **then** *lst1*
 else mmovn-*lst1* (*h*,
 movn-*lst* (*h*, *lst1*, *lst2*, $i - h$),
 lst2,
 $i - h$,
 $nt - 1$) **endif**

THEOREM: mmov1-*lst*-0

mmov1-*lst* (*i*, *lst1*, *lst2*, 0) = *lst1*

THEOREM: mmov1-*lst1*-0

mmov1-*lst1* (*i*, *lst1*, *lst2*, 0) = *lst1*

THEOREM: mmovn-*lst*-0

mmovn-*lst* (*n*, *lst1*, *lst2*, *i*, 0) = *lst1*

THEOREM: mmovn-*lst1*-0

mmovn-*lst1* (*n*, *lst1*, *lst2*, *i*, 0) = *lst1*

;MOD32-EQ

THEOREM: mod32-eq-deduction0

$\text{mod32-eq}(x, x)$

THEOREM: mod32-eq-deduction1

$(\text{mod32-eq}(0, \text{neg}(32, x)) = \text{mod32-eq}(x, 0))$

$\wedge (\text{mod32-eq}(\text{neg}(32, x), 0) = \text{mod32-eq}(x, 0))$

THEOREM: mod32-eq-deduction2

$(\text{mod32-eq}(x, \text{add}(32, x, y)) = \text{mod32-eq}(0, y))$

$\wedge (\text{mod32-eq}(\text{add}(32, x, y), x) = \text{mod32-eq}(y, 0))$

THEOREM: mod32-eq-deduction3

$\text{mod32-eq}(\text{add}(32, x, y), \text{add}(32, x, z)) = \text{mod32-eq}(y, z)$

EVENT: Disable mod32-eq.

; generate all the cases from between-ileq.

THEOREM: between-ileq-la

$(\text{integerp}(x) \wedge \text{integerp}(y) \wedge \text{integerp}(z))$

$\rightarrow (\text{between-ileq}(x, y, z)$

$= \text{if } \text{ilessp}(z, x) \text{ then f}$

$\text{else } (x = y) \vee \text{between-ileq}(\text{iplus}(x, 1), y, z) \text{ endif})$

EVENT: Disable between-ileq.

; readm-mem, mem-lst, and mem-ilst is not changed if read-mem is not
; changed.

THEOREM: read-mem-plus

$\text{read-mem}(\text{addr}, \text{mem}, m + k)$

$= \text{app}(8 * k, \text{read-mem}(\text{add}(32, \text{addr}, m), \text{mem}, k), \text{read-mem}(\text{addr}, \text{mem}, m))$

THEOREM: stepn-readm-mem

$(\text{read-mem}(x, \text{mc-mem}(\text{stepn}(s, n)), \text{opsz} * k)$

$= \text{read-mem}(x, \text{mc-mem}(s), \text{opsz} * k)$

$\rightarrow (\text{readm-mem}(\text{opsz}, x, \text{mc-mem}(\text{stepn}(s, n)), k)$

$= \text{readm-mem}(\text{opsz}, x, \text{mc-mem}(s), k))$

THEOREM: stepn-mem-lst

$(\text{readm-mem}(\text{opsz}, x, \text{mc-mem}(\text{stepn}(s, n)), k) = \text{readm-mem}(\text{opsz}, x, \text{mc-mem}(s), k))$

$\rightarrow (\text{mem-lst}(\text{opsz}, x, \text{mc-mem}(\text{stepn}(s, n)), k, \text{lst})$

$= \text{mem-lst}(\text{opsz}, x, \text{mc-mem}(s), k, \text{lst}))$

THEOREM: stepn-mem-ilst
 $(\text{readm-mem}(\text{opsz}, x, \text{mc-mem}(\text{stepn}(s, n)), k) = \text{readm-mem}(\text{opsz}, x, \text{mc-mem}(s), k))$
 $\rightarrow (\text{mem-ilst}(\text{opsz}, x, \text{mc-mem}(\text{stepn}(s, n)), k, \text{lst})$
 $= \text{mem-ilst}(\text{opsz}, x, \text{mc-mem}(s), k, \text{lst}))$

EVENT: Disable read-mem-plus.

```
;
;          DISABLE EVENTS
; many events are no longer useful upon the completion of this system.
; before we enter the verification phase, we simply disable them.
```

```
;;;;;;;;; disable arithmetic events.
```

EVENT: Disable plus-add1-1.

EVENT: Disable remainder-exit.

EVENT: Disable quotient-exit.

```
;;;;;;;;; disable definitions in the specification.
; operations: The semantics of these operations has been established as
; rewrite rules in the library, which will be triggered to apply when
; nat-to-uint or nat-to-int are presented.
```

EVENT: Disable mulu.

EVENT: Disable muls.

EVENT: Disable quot.

EVENT: Disable rem.

EVENT: Disable iquot.

EVENT: Disable irem.

EVENT: Disable lsl.

EVENT: Disable lsr.

EVENT: Disable asl.

EVENT: Disable asr.

; condition codes: To avoid to open up these flag definitions is one of the
; several efforts to keep the proving space managable. Another point is
; that one flag might have different semantics in different situations.

EVENT: Disable fix-bit.

EVENT: Disable add-v.

EVENT: Disable add-z.

EVENT: Disable add-n.

EVENT: Disable addx-v.

EVENT: Disable addx-z.

EVENT: Disable addx-n.

EVENT: Disable sub-v.

EVENT: Disable sub-n.

EVENT: Disable subx-v.

EVENT: Disable subx-z.

EVENT: Disable subx-n.

EVENT: Disable and-z.

EVENT: Disable and-n.

EVENT: Disable mulu-v.

EVENT: Disable mulu-z.

EVENT: Disable mulu-n.

EVENT: Disable muls-v.

EVENT: Disable muls-z.

EVENT: Disable muls-n.

EVENT: Disable or-z.

EVENT: Disable or-n.

EVENT: Disable divs-z.

EVENT: Disable divs-n.

EVENT: Disable divu-z.

EVENT: Disable divu-n.

EVENT: Disable rol-c.

EVENT: Disable rol-z.

EVENT: Disable rol-n.

EVENT: Disable ror-c.

EVENT: Disable ror-z.

EVENT: Disable ror-n.

EVENT: Disable lsl-c.

EVENT: Disable lsl-z.

EVENT: Disable lsl-n.

EVENT: Disable lsr-c.

EVENT: Disable lsr-z.

EVENT: Disable lsr-n.

EVENT: Disable asl-c.

EVENT: Disable asl-v.

EVENT: Disable asl-z.

EVENT: Disable asl-n.

EVENT: Disable asr-c.

EVENT: Disable asr-z.

EVENT: Disable asr-n.

EVENT: Disable roxl-c.

EVENT: Disable roxl-z.

EVENT: Disable roxl-n.

EVENT: Disable roxr-c.

EVENT: Disable roxr-z.

EVENT: Disable roxr-n.

EVENT: Disable move-z.

EVENT: Disable move-n.

EVENT: Disable ext-z.

EVENT: Disable ext-n.

EVENT: Disable swap-z.

EVENT: Disable swap-n.

EVENT: Disable not-z.

EVENT: Disable not-n.

EVENT: Disable eor-z.

EVENT: Disable eor-n.

; read/write defns: for each of them, we have had a set of rewrite rules.
; their definitions are not needed to open up.

EVENT: Disable pc-read-memp.

EVENT: Disable pc-read-mem.

EVENT: Disable read-memp.

EVENT: Disable read-mem.

EVENT: Disable write-memp.

EVENT: Disable write-mem.

EVENT: Disable readm-rn.

EVENT: Disable writem-rn.

```
;;;;;;;;; disable intermediate lemmas for read/write events:  to prove  
; some important theorems about read/write,  we proved many intermediate  
; lemmas.  It is time to discard them.
```

EVENT: Disable pc-read-memp-la0.

EVENT: Disable pc-read-memp-la1.

EVENT: Disable pc-read-memp-la2.

EVENT: Disable pc-read-memp-la3.

EVENT: Disable write-memp-la0.

EVENT: Disable write-memp-la1.

EVENT: Disable write-memp-la2.

EVENT: Disable write-memp-la3.

```
;;;;;;;;; disable intermediate lemmas for disjoint0/disjoint events.
```

EVENT: Disable disjoint0.

EVENT: Disable disjoint.

EVENT: Disable disjoint0-la0.

EVENT: Disable disjoint0-la1.

EVENT: Disable disjoint0-la2.

EVENT: Disable disjoint-la0.

EVENT: Disable disjoint-la1.

EVENT: Disable disjoint-la2.

EVENT: Disable disjoint-la3.

;;;;;;;;; disable intermediate lemmas for add/sub/neg events.

EVENT: Disable add-non-numberp.

EVENT: Disable add-commutativity.

EVENT: Disable add-commutativity1.

;;;;;;;;; disable mapping functions.

EVENT: Disable nat-to-uint.

EVENT: Disable uint-to-nat.

;;;;;;;;; disable miscellaneous lemmas.

EVENT: Disable stepn.

; addressing mode checking for each instruction. Their arguments are
; always "concrete" values. For all the instructions, the opln and ins
; have to become known at the execution time. Theoretically, it would
; be nice to disable them. But we do not want disabling to slow down
; the prover. Therefore, we only disable them when we feel necessary.

EVENT: Let us define the theory *mode-guards* to consist of the following events:
add-addr-modep1, add-addr-modep2, adda-addr-modep, sub-addr-modep1, sub-
addr-modep2, suba-addr-modep, and-addr-modep1, and-addr-modep2, mul&div-
addr-modep, or-addr-modep1, or-addr-modep2, s&r-addr-modep, move-addr-
modep, movea-addr-modep, lea-addr-modep, clr-addr-modep, move-from-ccr-
addr-modep, negx-addr-modep, neg-addr-modep, move-to-ccr-addr-modep, pea-
addr-modep, movem-rn-ea-addr-modep, movem-ea-rn-addr-modep, tst-addr-modep,
tas-addr-modep, jmp-addr-modep, jsr-addr-modep, not-addr-modep, scc-addr-
modep, addq-addr-modep, subq-addr-modep, cmp-addr-modep, cmpa-addr-modep,
eor&eorl-addr-modep, bchg-addr-modep, bclr-addr-modep, bset-addr-modep,

btst-addr-modep, ori-addr-modep, andi-addr-modep, subi-addr-modep, addi-addr-modep, cmpi-addr-modep.

; the classification of instructions, according to the opcode.

EVENT: Let us define the theory *groups* to consist of the following events: bit-group, move-ins, move-group, misc-group, scc-group, bcc-group, or-group, sub-group, cmp-group, and-group, add-group, s&r-group.

; INVARIANTS OF THE SPEC

CONSERVATIVE AXIOM: h-invariant

$$\begin{aligned} & (p(x, \text{write-mem}(value, y, mem, m), k) = p(x, mem, k)) \\ \wedge \quad & ((p(x, mem, k) \wedge \text{write-memp}(y, mem, m)) \\ & \rightarrow (h(x, \text{write-mem}(value, y, mem, m), k) = h(x, mem, k))) \end{aligned}$$

Simultaneously, we introduce the new function symbols p and h .

THEOREM: addr-index2-mem

$$\text{mc-mem}(\text{car}(\text{addr-index2}(pc, addr, indexwd, s))) = \text{mc-mem}(s)$$

THEOREM: immediate-mem

$$\text{mc-mem}(\text{car}(\text{immediate}(oplen, pc, s))) = \text{mc-mem}(s)$$

THEOREM: effec-addr-mem

$$\text{mc-mem}(\text{car}(\text{effec-addr}(oplen, mode, rn, s))) = \text{mc-mem}(s)$$

THEOREM: mc-instate-mem

$$\text{mc-mem}(\text{car}(\text{mc-instate}(oplen, ins, s))) = \text{mc-mem}(s)$$

THEOREM: mapping-h

$$\begin{aligned} & p(x, \text{mc-mem}(\text{car}(s\mathcal{E}addr)), k) \\ \rightarrow \quad & (h(x, \text{mc-mem}(\text{mapping}(oplen, v\mathcal{E}cvznx, s\mathcal{E}addr)), k) \\ & = h(x, \text{mc-mem}(\text{car}(s\mathcal{E}addr)), k)) \end{aligned}$$

DEFINITION: $t3(x, y, z) = t$

; add-group.

THEOREM: add-group-h

$$\begin{aligned} & p(x, \text{mc-mem}(s), k) \\ \rightarrow \quad & (h(x, \text{mc-mem}(\text{add-group}(opmode, ins, s)), k) = h(x, \text{mc-mem}(s), k)) \end{aligned}$$

; sub-group.

THEOREM: sub-group-h

$$\begin{aligned} & p(x, \text{mc-mem}(s), k) \\ \rightarrow & (h(x, \text{mc-mem}(\text{sub-group}(\text{opmode}, \text{ins}, s)), k) = h(x, \text{mc-mem}(s), k)) \end{aligned}$$

; and-group.

THEOREM: and-group-h

$$\begin{aligned} & p(x, \text{mc-mem}(s), k) \\ \rightarrow & (h(x, \text{mc-mem}(\text{and-group}(\text{oplen}, \text{ins}, s)), k) = h(x, \text{mc-mem}(s), k)) \end{aligned}$$

; or-group.

THEOREM: or-group-h

$$\begin{aligned} & p(x, \text{mc-mem}(s), k) \\ \rightarrow & (h(x, \text{mc-mem}(\text{or-group}(\text{oplen}, \text{ins}, s)), k) = h(x, \text{mc-mem}(s), k)) \end{aligned}$$

; scc-group.

THEOREM: scc-group-h

$$p(x, \text{mc-mem}(s), k) \rightarrow (h(x, \text{mc-mem}(\text{scc-group}(\text{ins}, s)), k) = h(x, \text{mc-mem}(s), k))$$

; cmp-group.

THEOREM: cmp-group-h

$$\begin{aligned} & p(x, \text{mc-mem}(s), k) \\ \rightarrow & (h(x, \text{mc-mem}(\text{cmp-group}(\text{oplen}, \text{ins}, s)), k) = h(x, \text{mc-mem}(s), k)) \end{aligned}$$

; bcc-group.

THEOREM: bcc-group-h

$$\begin{aligned} & p(x, \text{mc-mem}(s), k) \\ \rightarrow & (h(x, \text{mc-mem}(\text{bcc-group}(\text{disp}, \text{ins}, s)), k) = h(x, \text{mc-mem}(s), k)) \end{aligned}$$

; bit-group.

THEOREM: write-mem-maintain-movep-writep

$$\text{movep-writep}(\text{addr}, \text{write-mem}(\text{value}, x, \text{mem}, m), n) = \text{movep-writep}(\text{addr}, \text{mem}, n)$$

THEOREM: movep-write-h

$$\begin{aligned} & (p(x, \text{mem}, k) \wedge \text{movep-writep}(\text{addr}, \text{mem}, n)) \\ \rightarrow & (h(x, \text{movep-write}(\text{value}, \text{addr}, \text{mem}, n), k) = h(x, \text{mem}, k)) \end{aligned}$$

THEOREM: bit-group-h

$$p(x, \text{mc-mem}(s), k) \rightarrow (h(x, \text{mc-mem}(\text{bit-group}(\text{ins}, s)), k) = h(x, \text{mc-mem}(s), k))$$

; move-group.

THEOREM: move-h

$$\begin{aligned} & p(x, \text{mc-mem}(s), k) \\ \rightarrow & (h(x, \text{mc-mem}(\text{move-ins}(o\text{plen}, ins, s)), k) = h(x, \text{mc-mem}(s), k)) \end{aligned}$$

THEOREM: move-group-h

$$\begin{aligned} & p(x, \text{mc-mem}(s), k) \\ \rightarrow & (h(x, \text{mc-mem}(\text{move-group}(o\text{plen}, ins, s)), k) = h(x, \text{mc-mem}(s), k)) \end{aligned}$$

; s&r-group.

THEOREM: s&r-group-h

$$p(x, \text{mc-mem}(s), k) \rightarrow (h(x, \text{mc-mem}(\text{s\&r-group}(ins, s)), k) = h(x, \text{mc-mem}(s), k))$$

; misc-group.

THEOREM: writem-mem-h

$$\begin{aligned} & (p(x, mem, k) \wedge \text{write-memp}(addr, mem, opsz * \text{len}(v\text{lst}))) \\ \rightarrow & (h(x, \text{writem-mem}(opsz, v\text{lst}, addr, mem), k) = h(x, mem, k)) \end{aligned}$$

THEOREM: movem-rnlst-len

$$\text{len}(\text{movem-rnlst}(mask, i)) = \text{movem-len}(mask)$$

THEOREM: movem-pre-rnlst-len

$$\text{len}(\text{movem-pre-rnlst}(mask, i, lst)) = (\text{movem-len}(mask) + \text{len}(lst))$$

THEOREM: misc-group-h

$$\begin{aligned} & p(x, \text{mc-mem}(s), k) \\ \rightarrow & (h(x, \text{mc-mem}(\text{misc-group}(ins, s)), k) = h(x, \text{mc-mem}(s), k)) \end{aligned}$$

THEOREM: stepi-h

$$p(x, \text{mc-mem}(s), k) \rightarrow (h(x, \text{mc-mem}(\text{stepi}(s)), k) = h(x, \text{mc-mem}(s), k))$$

THEOREM: stepi-p

$$t3(x, \text{mc-mem}(s), k) \rightarrow (p(x, \text{mc-mem}(\text{stepi}(s)), k) = p(x, \text{mc-mem}(s), k))$$

THEOREM: stepn-h

$$p(x, \text{mc-mem}(s), k) \rightarrow (h(x, \text{mc-mem}(\text{stepn}(s, n)), k) = h(x, \text{mc-mem}(s), k))$$

; the instantiations.

THEOREM: stepn-pc-read-memp

$$\begin{aligned} & t3(x, \text{mc-mem}(s), k) \\ \rightarrow & (\text{pc-read-memp}(x, \text{mc-mem}(\text{stepn}(s, n)), k) = \text{pc-read-memp}(x, \text{mc-mem}(s), k)) \end{aligned}$$

THEOREM: stepn-read-memp

$$\begin{aligned} & t3(x, \text{mc-mem}(s), k) \\ \rightarrow & (\text{read-memp}(x, \text{mc-mem}(\text{stepn}(s, n)), k) = \text{read-memp}(x, \text{mc-mem}(s), k)) \end{aligned}$$

THEOREM: stepn-write-memp
 $t3(x, mc\text{-}mem(s), k)$
 $\rightarrow (write\text{-}memp(x, mc\text{-}mem(stepn(s, n)), k) = write\text{-}memp(x, mc\text{-}mem(s), k))$
; after the execution of n arbitrary instructions, ROM is still ROM.

THEOREM: stepn-rom-addrp
 $rom\text{-}addrp(x, mc\text{-}mem(stepn(s, n)), k) = rom\text{-}addrp(x, mc\text{-}mem(s), k)$
; after the execution of n arbitrary instructions, RAM is still RAM.

THEOREM: stepn-ram-addrp
 $ram\text{-}addrp(x, mc\text{-}mem(stepn(s, n)), k) = ram\text{-}addrp(x, mc\text{-}mem(s), k)$
; after the execution of n arbitrary instructions, the contents of
; the memory are not modified if that portion of the memory is ROM.

THEOREM: stepn-pc-read-mem
 $rom\text{-}addrp(x, mc\text{-}mem(s), k)$
 $\rightarrow (pc\text{-}read\text{-}mem(x, mc\text{-}mem(stepn(s, n)), k) = pc\text{-}read\text{-}mem(x, mc\text{-}mem(s), k))$

THEOREM: stepn-read-mem
 $rom\text{-}addrp(x, mc\text{-}mem(s), k)$
 $\rightarrow (read\text{-}mem(x, mc\text{-}mem(stepn(s, n)), k) = read\text{-}mem(x, mc\text{-}mem(s), k))$
; after the execution of n arbitrary instructions, the program segment
; maintains the same. Since we always require programs in ROM.

THEOREM: stepn-mcode-addrp
 $rom\text{-}addrp(x, mc\text{-}mem(s), len(lst))$
 $\rightarrow (mcode\text{-}addrp(x, mc\text{-}mem(stepn(s, n)), lst)$
 $\quad = mcode\text{-}addrp(x, mc\text{-}mem(s), lst))$

EVENT: Disable splus.

EVENT: Disable mcode-addrp.

; end of the proving phase. *****

; VERIFICATION PHASE

; our goal is to verify programs at machine code level. Before we go to
; specific programs, we introduce some conventions of machine code program
; constructs. This section provides some useful concepts for us to specify
; and verify machine code programs.
;
; look up as unsigned integers.

DEFINITION: $\text{uread-mem}(x, \text{mem}, n) = \text{nat-to-uint}(\text{read-mem}(x, \text{mem}, n))$

DEFINITION:

$\text{uread-dn}(\text{oplen}, \text{dn}, s) = \text{nat-to-uint}(\text{read-dn}(\text{oplen}, \text{dn}, s))$

DEFINITION:

$\text{uread-an}(\text{oplen}, \text{an}, s) = \text{nat-to-uint}(\text{read-an}(\text{oplen}, \text{an}, s))$

; look up as signed integers.

DEFINITION:

$\text{iread-mem}(x, \text{mem}, n) = \text{nat-to-int}(\text{read-mem}(x, \text{mem}, n), 8 * n)$

DEFINITION:

$\text{iread-dn}(\text{oplen}, \text{dn}, s) = \text{nat-to-int}(\text{read-dn}(\text{oplen}, \text{dn}, s), \text{oplen})$

DEFINITION:

$\text{iread-an}(\text{oplen}, \text{an}, s) = \text{nat-to-int}(\text{read-an}(\text{oplen}, \text{an}, s), \text{oplen})$

; the return address of subroutine call.

DEFINITION:

$\text{rts-addr}(s) = \text{read-mem}(\text{read-an}(32, 7, s), \text{mc-mem}(s), 4)$

DEFINITION:

$\text{linked-rts-addr}(s) = \text{read-mem}(\text{add}(32, \text{read-an}(32, 6, s), 4), \text{mc-mem}(s), 4)$

; the saved A6 on the stack.

DEFINITION:

$\text{linked-a6}(s) = \text{read-mem}(\text{read-an}(32, 6, s), \text{mc-mem}(s), 4)$

; for MOVEM instruction. After the link, A6 points to some place
; on the stack. s is the current machine state, opsz is the
; operation size of the MOVEM, i is the offset relative to a6,
; and n is the number of registers moved.

DEFINITION:

$\text{movem-saved}(s, \text{opsz}, i, n)$
 $= \text{readm-mem}(\text{opsz}, \text{sub}(32, i, \text{read-an}(32, 6, s)), \text{mc-mem}(s), n)$

; when only one register is saved, we use MOVE instead of MOVEM.

DEFINITION:

$\text{rn-saved}(s) = \text{read-mem}(\text{sub}(32, 4, \text{read-an}(32, 6, s)), \text{mc-mem}(s), 4)$

; We do not want to deal with nat-to-int and int-to-nat in the logic
; level. In the logic level, the proof concerns purely the algorithmic
; correctness in the problem. It is hoped to be machine independent.

DEFINITION:

lst-int (n , lst)
= **if** $lst \simeq \mathbf{nil}$ **then** lst
else cons (nat-to-int (car (lst), n), lst-int (n , cdr (lst))) **endif**

THEOREM: get-lst-int

get-nth (i , lst-int (n , lst)) = nat-to-int (get-nth (i , lst), n)

THEOREM: put-lst-int

put-nth (nat-to-int ($value$, n), i , lst-int (n , lst))
= lst-int (n , put-nth ($value$, i , lst))

DEFINITION:

lst-integerp (lst)
= **if** $lst \simeq \mathbf{nil}$ **then** **t**
else integerp (car (lst)) $\wedge$ lst-integerp (cdr (lst)) **endif**

THEOREM: get-lst-integerp

lst-integerp (lst) $\rightarrow$ integerp (get-nth (n , lst))

THEOREM: mem-lst-integerp

(mem-lst ($opsz$, a , mem , n , lst) $\wedge$ ($oplen = (8 * opsiz)$))
 $\rightarrow$ lst-integerp (lst-int ($oplen$, lst))

THEOREM: mem-ilst-lst-integerp

mem-ilst ($opsz$, a , mem , n , lst) $\rightarrow$ lst-integerp (lst)

EVENT: Disable mem-lst.

EVENT: Disable mem-ilst.

; a trick to get all the appearances of a term replaced by another
; term. For example, in subroutines, a6 is used as the addresss register
; for LINK. It is often relative to sp(a7). We'd like to have a6
; replaced by a7.

DEFINITION: equal* (x , y) = ($x = y$)

THEOREM: equal*-reflex

equal* (x , x)

THEOREM: read-rn-equal^*
 $\text{equal}^*(\text{read-rn}(\text{oplen}, rn, rfile), x) \rightarrow (\text{read-rn}(\text{oplen}, rn, rfile) = x)$

EVENT: Disable equal^* .

; the registers d0, d1, a0, and a1 are available for any subroutines.
; There is no obligation for subroutines to restore their previous values.
; Therefore, we do not need to keep track of them. We handle a7
; (the stack pointer) separately to make sure it is set correctly.
; a6 (the frame pointer) perhaps also deserves special treatment.

DEFINITION:
 $\text{d2-7a2-5p}(rn)$
 $= ((rn \neq 0)$
 $\quad \wedge (rn \neq 1)$
 $\quad \wedge (rn \neq 8)$
 $\quad \wedge (rn \neq 9)$
 $\quad \wedge (rn \neq 14)$
 $\quad \wedge (rn \neq 15))$

; d2 will be used for some purpose.

DEFINITION: $\text{d3-7a2-5p}(rn) = (\text{d2-7a2-5p}(rn) \wedge (rn \neq 2))$

; d2 and d3 will be used for some purpose.

DEFINITION:
 $\text{d4-7a2-5p}(rn) = (\text{d2-7a2-5p}(rn) \wedge (rn \neq 2) \wedge (rn \neq 3))$

; d2, d3, and d4 will be used for some purpose.

DEFINITION:
 $\text{d5-7a2-5p}(rn)$
 $= (\text{d2-7a2-5p}(rn) \wedge (rn \neq 2) \wedge (rn \neq 3) \wedge (rn \neq 4))$

; a2 will be used for some purpose.

DEFINITION: $\text{d2-7a3-5p}(rn) = (\text{d2-7a2-5p}(rn) \wedge (rn \neq 10))$

DEFINITION:
 $\text{d4-7a4-5p}(rn) = (\text{d4-7a2-5p}(rn) \wedge (rn \neq 10) \wedge (rn \neq 11))$

DEFINITION:
 $\text{d4-7a5p}(rn)$
 $= (\text{d4-7a2-5p}(rn) \wedge (rn \neq 10) \wedge (rn \neq 11) \wedge (rn \neq 12))$

DEFINITION:

$d5-7a4-5p(rn)$
 $= (d4-7a2-5p(rn) \wedge (rn \neq 4) \wedge (rn \neq 10) \wedge (rn \neq 11))$

DEFINITION:

$d6-7a2-5p(rn) = (d4-7a2-5p(rn) \wedge (rn \neq 4) \wedge (rn \neq 5))$

; something I don't know where to put yet.
; the mean $(i+j)/2 < j$ iff $i < j$.

THEOREM: mean-lessp-lemma

$((i + j) < (2 * j)) = (i < j)$

THEOREM: mean-lessp

$((((i + j) \div 2) < j) = (i < j))$
 $\wedge (((j + i) \div 2) < j) = (i < j))$

; swap the i th and j th elements of the list.

DEFINITION:

$swap(i, j, lst) = put_nth(get_nth(i, lst), j, put_nth(get_nth(j, lst), i, lst))$

THEOREM: get-swap

$get_nth(k, swap(i, j, lst))$
 $=$ if $fix(k) = fix(i)$ then $get_nth(j, lst)$
elseif $fix(k) = fix(j)$ then $get_nth(i, lst)$
else $get_nth(k, lst)$ endif

EVENT: Disable get-nth.

EVENT: Disable put-nth.

; C conventions.

DEFINITION: $NULL = 0$

DEFINITION: $EOF = -1$

; the Nqthm counterparts of the Berkeley C string functions.
; memchr.

DEFINITION:

$memchr1(i, n, lst, ch)$
 $=$ if $get_nth(i, lst) = ch$ then $fix(i)$
elseif $(n - 1) = 0$ then f
else $memchr1(1 + i, n - 1, lst, ch)$ endif

DEFINITION:

```
memchr(n, lst, ch)  
=  if n = 0 then f  
    else memchr1(0, n, lst, ch) endif  
  
; memcmp.
```

DEFINITION:

```
memcmp1(i, n, lst1, lst2)  
=  if get-nth(i, lst1) = get-nth(i, lst2)  
    then if (n - 1) = 0 then 0  
        else memcmp1(1 + i, n - 1, lst1, lst2) endif  
    else idifference(get-nth(i, lst1), get-nth(i, lst2)) endif
```

DEFINITION:

```
memcmp(n, lst1, lst2)  
=  if n = 0 then 0  
    else memcmp1(0, n, lst1, lst2) endif  
  
; memset.
```

DEFINITION:

```
memset1(i, n, lst, ch)  
=  if (n - 1) = 0 then put-nth(ch, i, lst)  
    else memset1(1 + i, n - 1, put-nth(ch, i, lst), ch) endif
```

DEFINITION:

```
memset(n, lst, ch)  
=  if n = 0 then lst  
    else memset1(0, n, lst, ch) endif  
  
; strlen.
```

DEFINITION:

```
strlen(i, n, lst)  
=  if i < n  
    then if get-nth(i, lst) = NULL then i  
        else strlen(1 + i, n, lst) endif  
    else i endif  
  
; strcpy.
```

DEFINITION:

```
strcpy(i, lst1, n2, lst2)  
=  if i < n2  
    then if get-nth(i, lst2) = NULL then put-nth(NULL, i, lst1)  
        else strcpy(1 + i, put-nth(get-nth(i, lst2), i, lst1), n2, lst2) endif  
    else lst1 endif
```

; strcmp.

DEFINITION:

strcmp(*i*, *n1*, *lst1*, *lst2*)

= if *i* < *n1*

then if get-nth(*i*, *lst1*) = get-nth(*i*, *lst2*)

then if get-nth(*i*, *lst1*) = NULL then 0

else strcmp(1 + *i*, *n1*, *lst1*, *lst2*) endif

else idifference(get-nth(*i*, *lst1*), get-nth(*i*, *lst2*)) endif

else 0 endif

; strcoll.

DEFINITION: strcoll(*n1*, *lst1*, *lst2*) = strcmp(0, *n1*, *lst1*, *lst2*)

; strcat.

DEFINITION:

strcpy1(*i*, *lst1*, *j*, *n2*, *lst2*)

= if *j* < *n2*

then if get-nth(*j*, *lst2*) = NULL then put-nth(NULL, *i*, *lst1*)

else strcpy1(1 + *i*,

put-nth(get-nth(*j*, *lst2*), *i*, *lst1*),

1 + *j*,

n2,

lst2) endif

else *lst1* endif

DEFINITION:

strcat(*n1*, *lst1*, *n2*, *lst2*)

= if get-nth(0, *lst1*) = NULL then strcpy1(0, *lst1*, 0, *n2*, *lst2*)

else strcpy1(strlen(1, *n1*, *lst1*), *lst1*, 0, *n2*, *lst2*) endif

; strncat.

DEFINITION:

strcpy2(*i*, *lst1*, *j*, *n*, *lst2*)

= if get-nth(*j*, *lst2*) = NULL then put-nth(NULL, *i*, *lst1*)

elseif (*n* - 1) = 0

then put-nth(NULL, 1 + *i*, put-nth(get-nth(*j*, *lst2*), *i*, *lst1*))

else strcpy2(1 + *i*,

put-nth(get-nth(*j*, *lst2*), *i*, *lst1*),

1 + *j*,

n - 1,

lst2) endif

DEFINITION:

```
strncat(n1, lst1, n, lst2)  
=  if n = 0 then lst1  
    elseif get-nth(0, lst1) = NULL then strcpy2(0, lst1, 0, n, lst2)  
    else strcpy2(strlen(1, n1, lst1), lst1, 0, n, lst2) endif  
  
; strncmp.
```

DEFINITION:

```
strncmpl(i, n, lst1, lst2)  
=  if get-nth(i, lst1) = get-nth(i, lst2)  
    then if get-nth(i, lst1) = 0 then 0  
        elseif (n - 1) = 0 then 0  
        else strncmpl(1 + i, n - 1, lst1, lst2) endif  
    else idifference(get-nth(i, lst1), get-nth(i, lst2)) endif
```

DEFINITION:

```
strncmp(n, lst1, lst2)  
=  if n  $\simeq$  0 then 0  
    else strncmpl(0, n, lst1, lst2) endif  
  
; strncpy.
```

DEFINITION:

```
zero-list1(i, n, lst)  
=  if (n - 1) = 0 then lst  
    else zero-list1(1 + i, n - 1, put-nth(0, i, lst)) endif
```

DEFINITION:

```
zero-list(i, n, lst) = zero-list1(1 + i, n, put-nth(0, i, lst))
```

DEFINITION:

```
strncpy1(i, n, lst1, lst2)  
=  if get-nth(i, lst2) = 0 then zero-list(i, n, lst1)  
    elseif (n - 1) = 0 then put-nth(get-nth(i, lst2), i, lst1)  
    else strncpy1(1 + i, n - 1, put-nth(get-nth(i, lst2), i, lst1), lst2) endif
```

DEFINITION:

```
strncpy(n, lst1, lst2)  
=  if n  $\simeq$  0 then lst1  
    else strncpy1(0, n, lst1, lst2) endif  
  
; strchr.
```

DEFINITION:

```

strchr(i, n, lst, ch)
=  if i < n
    then if get-nth(i, lst) = ch then fix(i)
        elseif get-nth(i, lst) = 0 then f
        else strchr(1 + i, n, lst, ch) endif
    else f endif

```

; strcspn.

DEFINITION:

```

strcspn(i1, n1, lst1, n2, lst2)
=  if i1 < n1
    then if strchr(0, n2, lst2, get-nth(i1, lst1)) then fix(i1)
        else strcspn(1 + i1, n1, lst1, n2, lst2) endif
    else f endif

```

; strrchr.

DEFINITION:

```

strrchr(i, n, lst, ch, j)
=  if i < n
    then if get-nth(i, lst) = ch
        then if get-nth(i, lst) = 0 then fix(i)
            else strrchr(1 + i, n, lst, ch, fix(i)) endif
        elseif get-nth(i, lst) = 0 then j
        else strrchr(1 + i, n, lst, ch, j) endif
    else j endif

```

; strspn.

DEFINITION:

```

strchr1(i, n, lst, ch)
=  if i < n
    then if get-nth(i, lst) = 0 then f
        elseif get-nth(i, lst) = ch then fix(i)
        else strchr1(1 + i, n, lst, ch) endif
    else f endif

```

THEOREM: strchr1-bounds

$$(n \not\prec \text{strchr1}(i, n, \text{lst}, ch)) \\ \wedge (\text{strchr1}(i, n, \text{lst}, ch) \rightarrow (\text{strchr1}(i, n, \text{lst}, ch) \not\prec i))$$

THEOREM: strchr1-false-0

$$\neg \text{strchr1}(i, n, \text{lst}, 0)$$

DEFINITION:

```
strspn(i1, n1, lst1, n2, lst2)  
=  if i1 < n1  
    then if strchr1(0, n2, lst2, get-nth(i1, lst1))  
        then strspn(1 + i1, n1, lst1, n2, lst2)  
        else fix(i1) endif  
    else f endif
```

; strpbrk.

DEFINITION:

```
strpbrk(i1, n1, lst1, n2, lst2)  
=  if i1 < n1  
    then if get-nth(i1, lst1) = 0 then f  
        elseif strchr1(0, n2, lst2, get-nth(i1, lst1)) then fix(i1)  
        else strpbrk(1 + i1, n1, lst1, n2, lst2) endif  
    else f endif
```

; strstr.

DEFINITION:

```
strstr(i, n1, lst1, n2, lst2, len)  
=  if i < n1  
    then let j be strchr1(i, n1, lst1, get-nth(0, lst2))  
        in  
        if j ∈ N  
            then if strncmp(len, mcd(1 + j, lst1), mcd(1, lst2))  
                = 0 then j  
                else strstr(1 + j, n1, lst1, n2, lst2, len) endif  
            else f endif endlet  
    else f endif
```

DEFINITION:

```
strstr(n1, lst1, n2, lst2)  
=  if get-nth(0, lst2) = 0 then 0  
    else strstr(0, n1, lst1, n2, lst2, strlen(0, n2 - 1, mcd(1, lst2))) endif
```

; memmove.

DEFINITION:

```
memmove-1(lst1, lst2, i, nt, n)  
=  if (n mod 4) = 0 then mmovn-lst(4, lst1, lst2, i, nt)  
    else mmov1-lst(i + (4 * nt),  
        mmovn-lst(4, lst1, lst2, i, nt),  
        lst2,  
        n mod 4) endif
```

DEFINITION:

```
memmove-0(lst1, lst2, i, nt, n)
=  if n < 4 then mmov1-lst (i + nt, mmov1-lst (i, lst1, lst2, nt), lst2, n)
    else memmove-1 (mmov1-lst (i, lst1, lst2, nt),
                    lst2,
                    i + nt,
                    n ÷ 4,
                    n) endif
```

DEFINITION:

```
memmove-4(lst1, lst2, i, nt, n)
=  if (n mod 4) = 0 then mmovn-lst1 (4, lst1, lst2, i, nt)
    else mmov1-lst1 (i - (4 * nt),
                    mmovn-lst1 (4, lst1, lst2, i, nt),
                    lst2,
                    n mod 4) endif
```

DEFINITION:

```
memmove-3(lst1, lst2, i, nt, n)
=  if n < 4
    then mmov1-lst1 (i - nt, mmov1-lst1 (i, lst1, lst2, nt), lst2, n)
    else memmove-4 (mmov1-lst1 (i, lst1, lst2, nt),
                    lst2,
                    i - nt,
                    n ÷ 4,
                    n) endif
```

DEFINITION:

```
memmove(str1, str2, n, lst1, lst2)
=  if n ≈ 0 then lst1
    elseif nat-to-uint (str1) = nat-to-uint (str2) then lst2
    else let x1 be nat-to-uint (str1),
          x2 be nat-to-uint (str2)
    in
    if x1 < x2
    then if ((x1 mod 4) = 0)
        ∧ ((x2 mod 4) = 0)
    then if n < 4 then mmov1-lst (0, lst1, lst2, n)
        else memmove-1 (lst1,
                        lst2,
                        0,
                        n ÷ 4,
                        n) endif
    elseif ((x1 mod 4) = (x2 mod 4))
        ∧ (3 < n)
```

```

    then memmove-0 (lst1,
                    lst2,
                    0,
                    4 - (x1 mod 4),
                    (n + (x1 mod 4)) - 4)
    else memmove-0 (lst1, lst2, 0, n, 0) endif
else let y1 be n + x1,
        y2 be n + x2
in
  if ((y1 mod 4) = 0)
    ∧ ((y2 mod 4) = 0)
  then if n < 4
    then mmov1-lst1 (n,
                    lst1,
                    lst2,
                    n)
    else memmove-4 (lst1,
                    lst2,
                    n,
                    n ÷ 4,
                    n) endif
  elseif ((y1 mod 4)
         = (y2 mod 4))
    ∧ (4 < n)
  then memmove-3 (lst1,
                  lst2,
                  n,
                  y1 mod 4,
                  n - (y1 mod 4))
  else memmove-3 (lst1,
                  lst2,
                  n,
                  n,
                  0) endif endlet endif endlet endif

```

; dual events of string functions. They are for internal use only and
; hardly visible for users.

DEFINITION:

strlen* (i^* , i , n , lst)

= if $i < n$

then if get-nth (i , lst) = NULL then i^*

else strlen* (add (32, i^* , 1), 1 + i , n , lst) endif

else i^* endif

DEFINITION:

```
memchr*(i*, i, n, lst, ch)
=  if get-nth(i, lst) = ch then i*
    elseif (n - 1) = 0 then f
    else memchr*(add(32, i*, 1), 1 + i, n - 1, lst, ch) endif
```

DEFINITION:

```
strchr*(i*, i, n, lst, ch)
=  if i < n
    then if get-nth(i, lst) = ch then i*
        elseif get-nth(i, lst) = 0 then f
        else strchr*(add(32, i*, 1), 1 + i, n, lst, ch) endif
    else f endif
```

DEFINITION:

```
strchr1*(i*, i, n, lst, ch)
=  if i < n
    then if get-nth(i, lst) = 0 then f
        elseif get-nth(i, lst) = ch then i*
        else strchr1*(add(32, i*, 1), 1 + i, n, lst, ch) endif
    else f endif
```

DEFINITION:

```
strcspn*(i1*, i1, n1, lst1, n2, lst2)
=  if i1 < n1
    then if strchr(0, n2, lst2, get-nth(i1, lst1)) then i1*
        else strcspn*(add(32, i1*, 1), 1 + i1, n1, lst1, n2, lst2) endif
    else f endif
```

DEFINITION:

```
strchr*(i*, i, n, lst, ch, j*)
=  if i < n
    then if get-nth(i, lst) = ch
        then if get-nth(i, lst) = 0 then i*
            else strchr*(add(32, i*, 1), 1 + i, n, lst, ch, i*) endif
        elseif get-nth(i, lst) = 0 then j*
        else strchr*(add(32, i*, 1), 1 + i, n, lst, ch, j*) endif
    else j* endif
```

DEFINITION:

```
strpbrk*(i1*, i1, n1, lst1, n2, lst2)
=  if i1 < n1
    then if get-nth(i1, lst1) = 0 then f
        elseif strchr1(0, n2, lst2, get-nth(i1, lst1)) then i1*
        else strpbrk*(add(32, i1*, 1), 1 + i1, n1, lst1, n2, lst2) endif
    else f endif
```

DEFINITION:

```

strspn*(i1*, i1, n1, lst1, n2, lst2)
=  if i1 < n1
    then if strchr1(0, n2, lst2, get-nth(i1, lst1))
        then strspn*(add(32, i1*, 1), 1 + i1, n1, lst1, n2, lst2)
        else i1* endif
    else f endif

```

DEFINITION:

```

strstr1*(i*, i, n1, lst1, n2, lst2, len)
=  if i < n1
    then let j* be strchr1*(i*, i, n1, lst1, get-nth(0, lst2)),
            j be strchr1(i, n1, lst1, get-nth(0, lst2))
    in
    if j ∈ N
    then if strncmp(len, mcd(1 + j, lst1), mcd(1, lst2))
        = 0 then j*
        else strstr1*(add(32, j*, 1),
                        1 + j,
                        n1,
                        lst1,
                        n2,
                        lst2,
                        len) endif
    else f endif endlet
    else f endif

```

DEFINITION:

```

strstr*(n1, lst1, n2, lst2)
=  if get-nth(0, lst2) = 0 then 0
    else strstr1*(0,
                  0,
                  n1,
                  lst1,
                  n2,
                  lst2,
                  strlen(0, n2 - 1, mcd(1, lst2))) endif

```

```

; strtok.
; the location of the token.

```

DEFINITION:

```

strtok-tok(str1, last, n1, lst1, n2, lst2)
=  let i* be strspn*(0, 0, n1, lst1, n2, lst2),
    i be strspn(0, n1, lst1, n2, lst2)

```

```

in
if nat-to-uint(str1) = 0
then if nat-to-uint(last) = 0 then 0
      elseif get-nth(i, lst1) = 0 then 0
      else add(32, last, i*) endif
elseif get-nth(i, lst1) = 0 then 0
else add(32, str1, i*) endif endlet

; the new lst1.

```

DEFINITION:

```

strtok-lst0(i1, lst1, ch)
= if ch = 0 then lst1
  else put-nth(0, i1, lst1) endif

```

DEFINITION:

```

strtok-lst1(i1, n1, lst1, n2, lst2)
= let i be strcspn(i1, n1, lst1, n2, lst2)
  in
  strtok-lst0(i, lst1, get-nth(i, lst1)) endlet

```

DEFINITION:

```

strtok-lst2(i1, n1, lst1, n2, lst2)
= if get-nth(i1, lst1) = 0 then lst1
  else strtok-lst1(1 + i1, n1, lst1, n2, lst2) endif

```

DEFINITION:

```

strtok-lst(n1, lst1, n2, lst2)
= strtok-lst2(strspn(0, n1, lst1, n2, lst2), n1, lst1, n2, lst2)

; the new value of the static variable.

```

DEFINITION:

```

strtok-last0(str1, i1*, i1, n1, lst1, n2, lst2)
= let i* be strcspn*(i1*, i1, n1, lst1, n2, lst2),
  i be strcspn(i1, n1, lst1, n2, lst2)
  in
  if get-nth(i, lst1) = 0 then 0
  else add(32, str1, add(32, i*, 1)) endif endlet

```

DEFINITION:

```

strtok-last1(str1, i1*, i1, n1, lst1, n2, lst2)
= if get-nth(i1, lst1) = 0 then 0
  else strtok-last0(str1, add(32, i1*, 1), 1 + i1, n1, lst1, n2, lst2) endif

```

DEFINITION:

```

strtok-last (str1, last, n1, lst1, n2, lst2)
=  let i* be strspn* (0, 0, n1, lst1, n2, lst2),
    i be strspn (0, n1, lst1, n2, lst2)
    in
    if nat-to-uint (str1) = 0
    then if nat-to-uint (last) = 0 then last
         else strtok-last1 (last, i*, i, n1, lst1, n2, lst2) endif
    else strtok-last1 (str1, i*, i, n1, lst1, n2, lst2) endif endlet

; strxfm.

```

DEFINITION:

```

strxfm1 (i, lst1, lst2, n)
=  if get-nth (i, lst2) = 0 then put-nth (0, i, lst1)
    elseif (n - 1) = 0 then put-nth (0, i, lst1)
    else strxfm1 (1 + i, put-nth (get-nth (i, lst2), i, lst1), lst2, n - 1) endif

```

DEFINITION:

```

strxfm (lst1, lst2, n)
=  if n ≈ 0 then lst1
    else strxfm1 (0, lst1, lst2, n) endif

; a list of characters.

```

DEFINITION:

```

lst-of-chrp (lst)
=  if listp (lst)
    then (car (lst) ∈ ℕ)
         ∧ (car (lst) < 256)
         ∧ lst-of-chrp (cdr (lst))
    else t endif

; theorems about lst-of-chrp.

```

THEOREM: get-lst-of-chrp

```

lst-of-chrp (lst) → ((get-nth (i, lst) < 256) ∧ (get-nth (i, lst) ∈ ℕ))

```

THEOREM: put-lst-of-chrp

```

lst-of-chrp (lst)
→ (lst-of-chrp (put-nth (x, i, lst)) = ((x ∈ ℕ) ∧ (x < 256)))

```

#|

```

(prove-lemma lessp-read-mem-1 (rewrite)
  (lessp (read-mem x mem 1) 256)
  ((use (byte-read-nat-range (n 8))))

```

```

      (enable nat-range)))

(prove-lemma mem-lst-of-chrp (rewrite)
  (implies (mem-lst 1 x mem n lst)
    (lst-of-chrp lst))
  ((enable mem-lst)))
|#

```

EVENT: Disable lst-of-chrp.

; the predicate stringp.

DEFINITION:

```

slen(i, n, lst)
=  if i < n
    then if get-nth(i, lst) = NULL then fix(i)
        else slen(1 + i, n, lst) endif
    else fix(i) endif

```

DEFINITION: stringp(*i*, *n*, *lst*) = (slen(*i*, *n*, *lst*) < *n*)

; events for slen, which is part of stringp.

THEOREM: slen-ubound

$(i \leq n) \rightarrow (n \not< \text{slen}(i, n, \text{lst}))$

THEOREM: slen-lbound

$\text{slen}(i, n, \text{lst}) \not< i$

THEOREM: slen-01

```

(slen(i, 0, lst) = fix(i))
^  (slen(i, 1, lst)
=   if (i < 1) ^ (get-nth(0, lst) ≠ 0) then 1
    else fix(i) endif

```

THEOREM: slen-add1

```

slen(i, 1 + i, lst)
=  if get-nth(i, lst) = 0 then fix(i)
    else 1 + i endif

```

THEOREM: slen-put0

$\text{slen}(i, n, \text{put-nth}(0, i, \text{lst})) = \text{fix}(i)$

THEOREM: slen-put

$(j < i) \rightarrow (\text{slen}(i, n, \text{put-nth}(v, j, \text{lst})) = \text{slen}(i, n, \text{lst}))$

THEOREM: lessp-slen-mcdr
 $(\text{slen}(i + k, n1, lst1) < n1)$
 $\rightarrow ((\text{slen}(i, n, \text{mcdr}(k, lst1)) < (n1 - k)) = \mathbf{t})$

THEOREM: lessp-slen-mcdr-0
 $(\text{slen}(1 + i, n1, lst1) < n1)$
 $\rightarrow ((\text{slen}(i, n, \text{mcdr}(1, lst1)) < (n1 - 1)) = \mathbf{t})$

THEOREM: slen-rec
 $((\text{get-nth}(i, lst) \neq 0) \wedge (i < n))$
 $\rightarrow (\text{slen}(1 + i, n, lst) = \text{slen}(i, n, lst))$

EVENT: Disable slen.

; theorems about stringp.

THEOREM: stringp-la
 $(\text{stringp}(i, n, lst) \wedge \text{int-rangep}(n, nn) \wedge (\text{get-nth}(i, lst) \neq 0))$
 $\rightarrow \text{int-rangep}(1 + i, nn)$

; theorems about strchr.

THEOREM: strchr-bounds
 $(n \not\prec \text{strchr}(i, n, lst, ch))$
 $\wedge (\text{strchr}(i, n, lst, ch) \rightarrow (\text{strchr}(i, n, lst, ch) \not\prec i))$

THEOREM: memchr-bounds
 $((i + n) \not\prec \text{memchr1}(i, n, lst, ch))$
 $\wedge (\text{memchr1}(i, n, lst, ch) \rightarrow (\text{memchr1}(i, n, lst, ch) \not\prec i))$

THEOREM: strpbrk-bounds
 $(n1 \not\prec \text{strpbrk}(i1, n1, lst1, n2, lst2))$
 $\wedge (\text{strpbrk}(i1, n1, lst1, n2, lst2) \rightarrow (\text{strpbrk}(i1, n1, lst1, n2, lst2) \not\prec i1))$

THEOREM: strchr-bounds
 $(j \leq n) \rightarrow (n \not\prec \text{strchr}(i, n, lst, ch, j))$

THEOREM: strstr-bounds
 $(n1 \not\prec \text{strstr1}(i, n1, lst1, n2, lst2, len))$
 $\wedge (\text{strstr1}(i, n1, lst1, n2, lst2, len)$
 $\rightarrow (\text{strstr1}(i, n1, lst1, n2, lst2, len) \not\prec i))$

THEOREM: strspn-ubound
 $(n1 \not\prec 0) \rightarrow (\text{strspn}(i, n1, lst1, n2, lst2) < n1)$

THEOREM: strspn-bounds

$$(n1 \not\prec \text{strspn}(i, n1, lst1, n2, lst2)) \\ \wedge (\text{strspn}(i, n1, lst1, n2, lst2) \rightarrow (\text{strspn}(i, n1, lst1, n2, lst2) \not\prec i))$$

THEOREM: strcspn-bounds

$$(n1 \not\prec \text{strcspn}(i, n1, lst1, n2, lst2)) \\ \wedge (\text{strcspn}(i, n1, lst1, n2, lst2) \rightarrow (\text{strcspn}(i, n1, lst1, n2, lst2) \not\prec i))$$

; a lemma to establish nat-range-p.

THEOREM: nat-range-p-la

$$(\text{nat-to-uint}(x) < \exp(2, n)) \rightarrow \text{nat-range-p}(x, n)$$

EVENT: Disable nat-range-p-la.

; some useful lemmas. I do not know where to put them yet.

; I will put them in the right place.

THEOREM: disjoint-1-int

$$(\text{disjoint}(a, m, b, n) \\ \wedge (j \leq m) \\ \wedge ((\text{nat-to-int}(k, 32) + l) \leq n) \\ \wedge (\text{nat-to-int}(k, 32) \in \mathbf{N})) \\ \rightarrow \text{disjoint}(a, j, \text{add}(32, b, k), l))$$

THEOREM: disjoint-2-int

$$(\text{disjoint}(a, m, b, n) \\ \wedge ((\text{nat-to-int}(i, 32) + j) \leq m) \\ \wedge (l \leq n) \\ \wedge (\text{nat-to-int}(i, 32) \in \mathbf{N})) \\ \rightarrow \text{disjoint}(\text{add}(32, a, i), j, b, l))$$

THEOREM: disjoint-2-int

$$(\text{disjoint}(a, m, b, n) \\ \wedge ((\text{nat-to-int}(i, 32) + j) \leq m) \\ \wedge (l \leq n) \\ \wedge (\text{nat-to-int}(i, 32) \in \mathbf{N})) \\ \rightarrow \text{disjoint}(b, l, \text{add}(32, a, i), j))$$

THEOREM: disjoint-3-int

$$(\text{disjoint}(a, m, b, n) \\ \wedge ((\text{nat-to-int}(i, 32) + j) \leq m) \\ \wedge ((\text{nat-to-int}(k, 32) + l) \leq n) \\ \wedge (\text{nat-to-int}(i, 32) \in \mathbf{N}) \\ \wedge (\text{nat-to-int}(k, 32) \in \mathbf{N})) \\ \rightarrow \text{disjoint}(\text{add}(32, a, i), j, \text{add}(32, b, k), l))$$

THEOREM: read-memp-ram1-uint
 $(\text{ram-addrp}(addr, mem, k) \wedge ((\text{nat-to-uint}(i) + j) \leq k))$
 $\rightarrow \text{read-memp}(\text{add}(32, addr, i), mem, j)$

THEOREM: write-memp-ram1-uint
 $(\text{ram-addrp}(addr, mem, k) \wedge ((\text{nat-to-uint}(i) + j) \leq k))$
 $\rightarrow \text{write-memp}(\text{add}(32, addr, i), mem, j)$

THEOREM: disjoint-1-uint
 $(\text{disjoint}(a, m, b, n) \wedge (j \leq m) \wedge ((\text{nat-to-uint}(k) + l) \leq n))$
 $\rightarrow \text{disjoint}(a, j, \text{add}(32, b, k), l)$

THEOREM: disjoint-2-uint
 $(\text{disjoint}(a, m, b, n) \wedge ((\text{nat-to-uint}(i) + j) \leq m) \wedge (l \leq n))$
 $\rightarrow \text{disjoint}(\text{add}(32, a, i), j, b, l)$

THEOREM: disjoint-2~uint
 $(\text{disjoint}(a, m, b, n) \wedge ((\text{nat-to-uint}(i) + j) \leq m) \wedge (l \leq n))$
 $\rightarrow \text{disjoint}(b, l, \text{add}(32, a, i), j)$

THEOREM: disjoint-3-uint
 $(\text{disjoint}(a, m, b, n)$
 $\wedge ((\text{nat-to-uint}(i) + j) \leq m)$
 $\wedge ((\text{nat-to-uint}(k) + l) \leq n))$
 $\rightarrow \text{disjoint}(\text{add}(32, a, i), j, \text{add}(32, b, k), l)$

THEOREM: disjoint-5-uint
 $(\text{disjoint}(\text{add}(32, a, i), m, b, n)$
 $\wedge ((j + \text{index-n}(0, i)) \leq m)$
 $\wedge ((\text{nat-to-uint}(k) + l) \leq n))$
 $\rightarrow \text{disjoint}(a, j, \text{add}(32, b, k), l)$

THEOREM: disjoint-6-uint
 $(\text{disjoint}(\text{add}(32, a, i), m, b, n)$
 $\wedge ((j + \text{index-n}(i1, i)) \leq m)$
 $\wedge (l \leq n))$
 $\rightarrow \text{disjoint}(\text{add}(32, a, i1), j, b, l)$

THEOREM: disjoint-7-uint
 $(\text{disjoint}(\text{add}(32, a, i), m, b, n)$
 $\wedge ((j + \text{index-n}(i1, i)) \leq m)$
 $\wedge ((\text{nat-to-uint}(k) + l) \leq n))$
 $\rightarrow \text{disjoint}(\text{add}(32, a, i1), j, \text{add}(32, b, k), l)$

THEOREM: read-rn-int-8.32
 $\text{nat-to-int}(\text{read-rn}(8, rn, rfile), 32) = \text{nat-to-uint}(\text{read-rn}(8, rn, rfile))$

THEOREM: read-rn-int-16_32
 $\text{nat-to-int}(\text{read-rn}(16, rn, rfile), 32) = \text{nat-to-uint}(\text{read-rn}(16, rn, rfile))$

THEOREM: read-rn-int-8_16
 $\text{nat-to-int}(\text{read-rn}(8, rn, rfile), 16) = \text{nat-to-uint}(\text{read-rn}(8, rn, rfile))$

THEOREM: read-mem-int-8_32
 $\text{nat-to-int}(\text{read-mem}(x, mem, 1), 32) = \text{nat-to-uint}(\text{read-mem}(x, mem, 1))$

THEOREM: read-mem-int-16_32
 $\text{nat-to-int}(\text{read-mem}(x, mem, 2), 32) = \text{nat-to-uint}(\text{read-mem}(x, mem, 2))$

THEOREM: read-mem-int-8_16
 $\text{nat-to-int}(\text{read-mem}(x, mem, 1), 16) = \text{nat-to-uint}(\text{read-mem}(x, mem, 1))$

THEOREM: idifference-int-range
 $((x \in \mathbf{N}) \wedge (y \in \mathbf{N}) \wedge \text{int-range}(x, n) \wedge \text{int-range}(y, n))$
 $\rightarrow \text{int-range}(\text{idifference}(x, y), n)$

EVENT: Disable idifference.

; some more arithmetic.

THEOREM: difference-cancel-1
 $(i \leq j)$
 $\rightarrow (((i + j) - (2 * i)) = (j - i))$
 $\wedge (((2 * j) - (i + j)) = (j - i))$

THEOREM: difference-is-1
 $((x - y) = 1) = (x = (1 + y))$

THEOREM: mean-difference-1
 $(i \leq j) \rightarrow (((i + j) \div 2) - i) = ((j - i) \div 2)$

THEOREM: mean-difference-2
 $(i \leq j) \rightarrow (((j - i) \div 2) \not\leq ((j - 1) - ((i + j) \div 2)))$

THEOREM: plus-0
 $(0 + x) = \text{fix}(x)$

THEOREM: plus-times-sub1
 $(x + (x * (y - 1)))$
 $= \text{if } y \simeq 0 \text{ then } \text{fix}(x)$
 $\text{else } x * y \text{ endif}$

EVENT: Disable plus-times-sub1.

THEOREM: plus2-times-sub1

$(x + y + (x * (z - 1)))$
= **if** $z \simeq 0$ **then** $x + y$
 else $y + (x * z)$ **endif**

THEOREM: plus3-times-sub1

$(x + y + z + (x * (z1 - 1)))$
= **if** $z1 \simeq 0$ **then** $x + y + z$
 else $y + z + (x * z1)$ **endif**

THEOREM: lessp-cancel-4294967295

$((4294967295 + x) < 4294967296) = (x \simeq 0)$

THEOREM: difference-cancel-4294967295

$((4294967295 + x) - 4294967296) = (x - 1)$

THEOREM: lessp-cancel-4294967292

$((4294967292 + x) < 4294967296) = (x < 4)$

THEOREM: difference-cancel-4294967292

$((4294967292 + x) - 4294967296) = (x - 4)$

THEOREM: difference-lessp-cancel-1

$((a - c) < (c * (b - 1)))$
= **if** $c \leq a$ **then** $a < (c * b)$
 else $1 < b$ **endif**

THEOREM: difference-lessp-cancel-2

$(c \leq a)$
→ $((a - c) < (d + (c * (b - 1))))$
= **if** $b \simeq 0$ **then** $(a - c) < d$
 else $a < (d + (c * b))$ **endif**

; two funny lemmas. But seems more useful than add-uint!

THEOREM: add-uintxx

$(\text{nat-range}(x, n)$
 $\wedge \text{nat-range}(y, n)$
 $\wedge ((\text{nat-to-uint}(x) + \text{nat-to-uint}(y)) < \exp(2, n)))$
→ $(\text{nat-to-uint}(\text{add}(n, x, y)) = (\text{nat-to-uint}(x) + \text{nat-to-uint}(y)))$

THEOREM: add-uintxxx

$(\text{nat-range}(x, n)$
 $\wedge \text{nat-range}(y, n)$
 $\wedge ((\text{nat-to-uint}(x) + \text{nat-to-uint}(y)) \not< \exp(2, n)))$
→ $(\text{nat-to-uint}(\text{add}(n, x, y))$
 $= ((\text{nat-to-uint}(x) + \text{nat-to-uint}(y)) - \exp(2, n)))$

EVENT: Disable add-uintxxx.

THEOREM: ram-addrp-3

$$\begin{aligned} & (\text{ram-addrp}(\text{add}(32, \text{addr}, i), \text{mem}, k) \wedge ((\text{index-n}(h, i) + j) \leq k)) \\ \rightarrow & \text{ram-addrp}(\text{add}(32, \text{addr}, h), \text{mem}, j) \end{aligned}$$

THEOREM: ram-addrp-4

$$\begin{aligned} & (\text{ram-addrp}(\text{addr}, \text{mem}, k) \wedge ((\text{nat-to-uint}(h) + j) \leq k)) \\ \rightarrow & \text{ram-addrp}(\text{add}(32, \text{addr}, h), \text{mem}, j) \end{aligned}$$

THEOREM: and-z-commutativity

$$\text{and-z}(n, x, y) = \text{and-z}(n, y, x)$$

; finally, establish the database.

EVENT: Make the library "mc20-2" and compile it.

Index

abs, 17, 63
abs-lessp-int-range, 17
add, 2, 3, 7, 23–30, 48, 54, 55, 74–
88, 93–100, 102, 103, 106,
118, 128–131, 135, 136, 138,
139
add-0, 26
add-adder, 27
add-addx-c, 57
add-addx-v, 66
add-associativity, 26
add-bcs&cc, 65
add-bmi, 68
add-bmi-crock1, 14
add-bmi-crock2, 14
add-bvs&vc, 67
add-c, 49, 57, 65
add-c-bitp, 49
add-cancel, 28
add-cancel0, 28
add-commutativity, 26
add-commutativity1, 26
add-commutativity2, 28
add-equal-cancel-1, 55
add-evenp, 7
add-fringe, 54–57
add-group, 114
add-group-h, 114
add-head, 27
add-int, 30
add-leq, 26
add-n, 49, 68
add-n-bitp, 49
add-nat-la, 23
add-nat-range, 27
add-neg-0, 54
add-neg-adder, 27
add-neg-cancel, 54
add-non-numberp, 74
add-of-0, 25
add-plus, 74
add-sub, 28
add-sub-cancel, 27
add-tree, 54–57
add-tree-add-fringe, 55
add-tree-append, 55
add-tree-bagdiff, 55
add-tree-delete, 56
add-tree-delete-cond, 55, 56
add-tree-delete-equal, 55
add-tree-nat-range, 55
add-uint, 24
add-uintxx, 138
add-uintxxx, 138
add-v, 49, 66, 67
add-v-bitp, 49
add-z, 49
add-z-bitp, 49
adder, 7, 8, 24–29
adder-associativity, 25
adder-cancel, 28
adder-commutativity, 25
adder-commutativity1, 25
adder-head, 27
adder-int, 29
adder-int-bridge, 29
adder-int-end, 13, 29
adder-lognot, 8
adder-nat-la, 24
adder-nat-range, 26
adder-shift-carry, 25
adder-uint, 24
addr-index2, 114
addr-index2-mem, 114
addx-c, 50, 57, 58
addx-c-bitp, 50
addx-c-la, 58
addx-n, 50
addx-n-bitp, 50
addx-v, 50, 66
addx-v-bitp, 50
addx-v-crock1, 13

addx-v-crock2, 14
 addx-v-la, 66
 addx-z, 50
 addx-z-bitp, 50
 addy-y, 29
 addy-y1, 29
 and-group, 115
 and-group-h, 115
 and-n, 50
 and-n-bitp, 50
 and-z, 50, 61, 62, 139
 and-z-bitp, 50
 and-z-commutativity, 139
 app, 9–11, 32–34, 44, 46, 79, 90, 91,
 106
 app-0, 9
 app-associativity, 33
 app-cancel, 44
 app-head-tail, 33
 app-nat-range, 34
 append-len, 91
 asl, 10, 12, 86, 87, 98–100
 asl-0, 12
 asl-asl, 12
 asl-beq, 64
 asl-bmi, 69
 asl-bvs, 68
 asl-c, 52
 asl-c-bitp, 52
 asl-int, 10
 asl-int-crock1, 9
 asl-int-crock2, 10
 asl-n, 52, 70
 asl-n-bitp, 52
 asl-nat-range, 12
 asl-v, 52, 68
 asl-v-bitp, 52
 asl-z, 52, 64
 asl-z-bitp, 52
 asr, 11, 12
 asr-0, 12
 asr-beq, 64
 asr-bmi, 70
 asr-c, 52
 asr-c-bitp, 52
 asr-int, 11
 asr-int-crock, 11
 asr-n, 52, 70
 asr-n-bitp, 52
 asr-nat-range, 12
 asr-z, 52, 64
 asr-z-bitp, 52

 b, 49, 79
 b-and, 4, 6
 b-eor, 4, 6
 b-not, 3, 6, 8, 35, 59, 66, 70–72
 b-or, 4, 6
 b0, 3, 4, 6
 b0p, 3
 b1, 3
 bagdiff, 55, 56
 bagint, 56
 bcar, 2, 32, 33
 bcar-1, 33
 bcar-2, 33
 bcar-app, 33
 bcar-head, 33
 bcar-lessp, 33
 bcar-nonnumberp, 32
 bcar-replace, 33
 bcc-group, 115
 bcc-group-h, 115
 bcd, 2, 32, 33
 bcd-1, 33
 bcd-2, 33
 bcd-lessp, 33
 bcd-nonnumberp, 32
 bcs, 65
 beq, 59–64
 between-ileq, 58, 106
 between-ileq-la, 106
 bge, 70, 71
 bge-v0, 70
 bgt, 71, 72
 bhi, 58, 59
 bit, 3, 4, 6
 bit-group, 115

- bit-group-h, 115
- bitn, 7, 12, 34, 35, 43, 44, 49, 66
- bitn-0, 34
- bitn-0-1, 34
- bitn-app, 34
- bitn-bitp, 49
- bitn-head, 34
- bitn-lognot, 35
- bitn-replace, 34
- bitn-tail, 34
- bitp, 7, 8, 13–15, 24, 29, 30, 49–53, 58, 66
- bitp-fix-bit, 49
- ble, 71, 72
- ble-bgt, 72
- bls, 59
- bls-bhi, 59
- blt, 70, 71
- blt-bge, 71
- blt-v0, 70
- bmi, 68–70
- bv-adder, 4, 5, 7
- bv-adder-bridge, 7
- bv-adder-len, 5
- bv-adder-listp, 5
- bv-adder-nat, 7
- bv-adder&carry, 6, 7
- bv-adder&carry-len, 6
- bv-adder&carry-nat, 7
- bv-bitn, 4, 6, 7
- bv-bitn-bitn, 7
- bv-bitn-not, 6
- bv-head, 3, 5–7
- bv-head-len, 5
- bv-head-listp, 5
- bv-head-nat, 5
- bv-len, 3–7
- bv-len-listp, 4
- bv-mbit, 4, 7
- bv-mbit-bitn, 7
- bv-not, 3, 6, 7
- bv-not-lognot, 7
- bv-sized-to-nat, 4–6
- bv-sized-to-nat-head, 5
- bv-sized-to-nat-to-bv-sized, 6
- bv-tail, 3
- bv-to-lst, 91–93, 96
- bv-to-lst-len, 92
- bv-to-lst-proper-lstp, 92
- bv-to-nat, 4–7
- bv-to-nat-range, 6
- bv-to-nat-to-bv, 5
- bv-to-nat-to-bv-sized, 5
- bvs, 67, 68
- byte-read, 36, 38, 45, 48, 90
- byte-read-nat-range, 38
- byte-read-write, 45
- byte-read-write-mem, 48
- byte-read-write-mem-end, 48
- byte-read-write-mem-lemma, 48
- byte-read=read-mem-1, 90
- byte-readp, 36, 45, 46, 76, 78, 100
- byte-readp-ram0, 78
- byte-readp-ram1, 78
- byte-readp-ram2, 78
- byte-readp-ram3, 78
- byte-readp-rom0, 76
- byte-readp-rom1, 76
- byte-readp-rom2, 76
- byte-readp-rom3, 76
- byte-write, 36, 45–49, 73, 81, 90, 95, 96
- byte-write-app, 46
- byte-write-maintain-byte-readp, 46
- byte-write-maintain-byte-writep, 46
- byte-write-maintain-pc-byte-readp, 45
- byte-write-maintain-pc-read-memp, 47
- byte-write-maintain-ram-addrp, 73
- byte-write-maintain-read-memp, 47
- byte-write-maintain-rom-addrp, 73
- byte-write-maintain-write-memp, 47
- byte-write-mcode-addrp, 81
- byte-write-write, 45
- byte-write-write-la, 45
- byte-write=write-mem-1, 90

- byte-writep, 36, 45–47, 74, 77, 81, 90, 100
- byte-writep->readp, 45
- byte-writep-ram0, 77
- byte-writep-ram1, 77
- byte-writep-ram2, 77
- byte-writep-ram3, 77
- byte-writep=write-memp-1, 90
- cancel-add-neg, 57
- cancel-equal-add, 56
- ccr-c, 39
- ccr-n, 39
- ccr-v, 39
- ccr-x, 39
- ccr-z, 39
- cdr-mcdr, 92
- cmp-group, 115
- cmp-group-h, 115
- correctness-of-cancel-add-neg, 57
- correctness-of-cancel-equal-add, 56
- cvzmx, 39
- d2-7a2-5p, 120
- d2-7a3-5p, 120
- d3-7a2-5p, 120
- d4-7a2-5p, 120, 121
- d4-7a4-5p, 120
- d4-7a5p, 120
- d5-7a2-5p, 120
- d5-7a4-5p, 121
- d6-7a2-5p, 121
- delete, 55–57
- difference-app-1s, 11
- difference-cancel-1, 137
- difference-cancel-4294967292, 138
- difference-cancel-4294967295, 138
- difference-int-rangep, 17
- difference-is-1, 137
- difference-lessp-cancel-1, 138
- difference-lessp-cancel-2, 138
- difference-nat-rangep, 37
- disjoint, 3, 48, 81–88, 94, 95, 100, 101, 135, 136
- disjoint-0, 82
- disjoint-0;, 84
- disjoint-1, 82
- disjoint-1-int, 135
- disjoint-1-uint, 136
- disjoint-10, 83
- disjoint-10;, 85
- disjoint-11, 83
- disjoint-11-asl, 87
- disjoint-11;, 85
- disjoint-11~asl, 87
- disjoint-12, 83
- disjoint-12;, 85
- disjoint-13, 83
- disjoint-13;, 85
- disjoint-14, 84
- disjoint-14;, 85
- disjoint-15, 84
- disjoint-15;, 85
- disjoint-1;, 84
- disjoint-2, 82
- disjoint-2-asl, 86
- disjoint-2-int, 135
- disjoint-2-uint, 136
- disjoint-2;, 84
- disjoint-2~asl, 86
- disjoint-2~int, 135
- disjoint-2~uint, 136
- disjoint-3, 82
- disjoint-3-asl, 86
- disjoint-3-int, 135
- disjoint-3-uint, 136
- disjoint-3;, 84
- disjoint-3~asl, 86
- disjoint-4, 82
- disjoint-4;, 84
- disjoint-5, 82
- disjoint-5-uint, 136
- disjoint-5;, 84
- disjoint-6, 82
- disjoint-6-uint, 136
- disjoint-6;, 84
- disjoint-7, 83
- disjoint-7-uint, 136

disjoint-7, 85
 disjoint-8, 83
 disjoint-8, 85
 disjoint-9, 83
 disjoint-9-asl, 86
 disjoint-9, 85
 disjoint-9-asl, 87
 disjoint-commutativity, 84
 disjoint-deduction0, 87
 disjoint-deduction1, 88
 disjoint-deduction2, 88
 disjoint-deduction3, 88
 disjoint-head, 81
 disjoint-la0, 82
 disjoint-la1, 82
 disjoint-la2, 82
 disjoint-la3, 82
 disjoint-leq, 95
 disjoint-leq-uint, 101
 disjoint-leq1, 95
 disjoint-leq1-uint, 101
 disjoint-x-x, 87
 disjoint0, 2, 3, 48, 49, 81, 82, 87, 95
 disjoint0-deduction0, 87
 disjoint0-deduction1, 87
 disjoint0-deduction2, 87
 disjoint0-head, 81
 disjoint0-la0, 81
 disjoint0-la1, 81
 disjoint0-la2, 81
 disjoint0-leq, 95
 disjoint0-x-x, 87
 divs-beq, 63
 divs-n, 51
 divs-n-bitp, 51
 divs-overflow, 23
 divs-v, 23
 divs-z, 51, 63
 divs-z-bitp, 51
 divs_3232-overflow, 23
 divu-beq, 63
 divu-n, 51
 divu-n-bitp, 51
 divu-overflow, 23
 divu-v, 23
 divu-z, 51, 63
 divu-z-bitp, 51
 effec-addr, 114
 effec-addr-mem, 114
 eof, 121
 eor-n, 53
 eor-n-bitp, 53
 eor-z, 53, 61
 eor-z-bitp, 53
 equal*, 119, 120
 equal*-reflex, 119
 eval\$-add-member, 55
 evenp, 7
 exp, 5, 6, 8–15, 17, 21–25, 29–31,
 35, 37, 58, 61–65, 67, 86,
 87, 89, 98, 99, 135, 138
 exp2-leq, 31
 exp2-lessp, 12
 exp2-lessp-crock, 21
 ext, 12, 38, 45, 60
 ext-0, 38
 ext-beq-int-0, 64
 ext-beq-int-1, 64
 ext-beq-uint, 64
 ext-bmi, 70
 ext-equal, 45
 ext-equal-0, 45
 ext-int, 12
 ext-lemma, 38
 ext-n, 53, 70
 ext-n-bitp, 53
 ext-nat-range, 38
 ext-z, 53, 64
 ext-z-bitp, 53
 fix-bit, 6, 7, 13, 39, 49
 fix-bit-bitp, 49
 fix-bv, 3, 6
 fix-bv-adder&carry, 6
 fix-int, 16–20
 fix-int-integerp, 18
 get-lst, 90, 92, 93, 104

- get-lst-cdr, 92
- get-lst-int, 119
- get-lst-integerp, 119
- get-lst-mcar, 92
- get-lst-mcdr, 92
- get-lst-of-chrp, 132
- get-nth, 40, 41, 79, 80, 89–94, 98, 103–105, 119, 121–126, 128–134
- get-nth-0, 40
- get-nth-append, 93
- get-nth-nil, 41
- get-nth-tail-lst, 80
- get-put, 41
- get-swap, 121
- get-vals, 90–93, 96, 104, 105
- get-vals-0, 96
- get-vals-append, 93
- get-vals-cdr, 92
- get-vals-len, 91
- get-vals-mcar, 92
- get-vals-mcdr, 92
- get-vals-proper-lstp, 92
- get-vlst, 42, 43
- get-vlst-member, 42
- get-vlst-readm-rn, 43
- groups, 114
- h, 114–116
- h-invariant, 114
- head, 2, 8–12, 21, 26–28, 31–38, 41, 42, 44, 45, 48, 53–55, 74, 81, 88–91, 95, 96, 101
- head-0, 8
- head-app, 32
- head-app-cancel, 44
- head-app-head-tail, 33
- head-byte-pc-read, 36
- head-byte-read, 36
- head-byte-readp, 36
- head-byte-write, 36
- head-byte-writep, 36
- head-ext, 38
- head-head, 31
- head-int-crock, 21
- head-lemma, 9
- head-leq, 8
- head-lessp, 8
- head-mem-ilst, 89
- head-mem-lst, 89
- head-nat-range, 37
- head-of-0, 8
- head-pc-byte-readp, 36
- head-pc-read, 36
- head-pc-read-mem, 36
- head-pc-read-memp, 36
- head-pc-readp, 35
- head-plus-cancel, 31
- head-plus-cancel0, 31
- head-plus-head, 31
- head-read, 36
- head-read-mem, 36
- head-read-memp, 36
- head-read-rn, 41
- head-readm-mem, 37
- head-readp, 35
- head-recursion, 44
- head-replace, 32
- head-sub1-lessp, 48
- head-write, 36
- head-write-mem, 36
- head-write-memp, 36
- head-writep, 36
- idifference, 15, 18, 19, 30, 66–68, 122–124, 137
- idifference-int-range, 137
- idifference-int-range1, 15
- idifference-int-range2, 15
- idifference-integerp, 18
- idifference-negativep, 19
- idifference-x-x, 19
- ileq, 58
- ilessp, 14, 15, 19, 68, 70, 71, 106
- ilessp-crock1, 14
- ilessp-crock2, 14
- ilessp-entails-ileq, 19
- ilessp-reflex, 19

immediate, 114
 immediate-mem, 114
 index-n, 75–88, 136, 139
 index-n-0, 88
 index-n-deduction0, 88
 index-n-deduction1, 88
 index-n-deduction2, 88
 index-n-x-x, 88
 ineg, 8, 18, 20
 ineg-integerp, 18
 ineg-iplus, 18
 int-range, 10, 13–17, 21–23, 29, 30, 62–64, 66–69, 89, 134, 137
 int-range-la, 15
 int-range-of-0, 17
 int-to-nat, 13, 16, 21
 int-to-nat-0, 16
 int-to-nat-range, 16
 int-to-nat-to-int, 16
 int-to-nat=0, 16
 integerp, 13–19, 21, 63, 89, 106, 119
 integerp-fix-int, 17
 integerp-minus0, 18
 iplus, 8, 11, 13–15, 18, 19, 29, 30, 66–68, 106
 iplus-0, 19
 iplus-1-1, 19
 iplus-associativity, 18
 iplus-commutativity, 18
 iplus-commutativity1, 18
 iplus-int-range1, 14
 iplus-int-range2, 15
 iplus-integerp, 18
 iplus-with-carry-negativep, 13
 iplus-with-carry-non-negativep, 13
 iquot, 22, 38
 iquot-int, 22
 iquot-nat-range, 38
 iquotient, 11, 17, 18, 20, 22, 23, 63
 iquotient-int-range, 17
 iquotient-integerp, 18
 iquotient-wrt-1, 20
 iquotient-wrt-1, 20
 iquotient=0, 63
 iread-an, 118
 iread-dn, 118
 iread-mem, 118
 iread-mem-get0, 94
 irem, 22, 38
 irem-int, 22
 irem-nat-range, 38
 iremainder, 11, 17–20, 22
 iremainder-int-range, 17
 iremainder-integerp, 18
 iremainder-wrt-1, 20
 iremainder-wrt-1, 19
 itimes, 10, 18, 19, 21, 62, 67, 69
 itimes-0, 19
 itimes-associativity, 19
 itimes-commutativity, 19
 itimes-equal-0, 19
 itimes-equal-cancellation, 19
 itimes-integerp, 18
 itimes-sign, 19
 izerop, 19
 l, 88
 len, 41, 42, 79–81, 91–93, 96, 100–102, 116, 117
 lessp-app-1s, 10
 lessp-cancel-4294967292, 138
 lessp-cancel-4294967295, 138
 lessp-slen-mcdr, 134
 lessp-slen-mcdr-0, 134
 lessp-times-exp-1s, 10
 linked-a6, 118
 linked-rtts-addr, 118
 log, 35, 61, 62
 logand, 35, 37
 logand-beq-uint, 61
 logand-commutativity, 35
 logand-eor-beq-uint, 62
 logand-logeor, 35
 logand-logor, 35
 logand-nat-range, 37
 logand-or-beq-uint, 61
 logand-uint, 35
 logand-uint-la, 35

logeor, 35, 37, 62
 logeor-beq-int-0, 61
 logeor-beq-int-1, 61
 logeor-beq-uint, 61
 logeor-commutativity, 35
 logeor-equal-0, 35
 logeor-nat-range, 37
 lognot, 7, 8, 27, 35, 66
 lognot-0, 7
 lognot-cancel, 8
 lognot-int, 8
 lognot-lognot, 8
 lognot-nat-range, 7
 logor, 35, 37, 61
 logor-beq-int-0, 61
 logor-beq-int-1, 61
 logor-beq-uint, 60
 logor-commutativity, 35
 logor-equal-0, 35
 logor-nat-range, 37
 lsl, 9, 11
 lsl-0, 11
 lsl-1-bcs&cc, 65
 lsl-beq, 63
 lsl-c, 52, 65
 lsl-c-0, 65
 lsl-c-bitp, 52
 lsl-lsl, 11
 lsl-n, 52
 lsl-n-bitp, 52
 lsl-nat-range, 11
 lsl-uint, 9
 lsl-z, 52, 63
 lsl-z-bitp, 52
 lsr, 9, 11, 12
 lsr-0, 11
 lsr-1-bcs&cc, 65
 lsr-beq, 63
 lsr-c, 52, 65
 lsr-c-0, 65
 lsr-c-bitp, 52
 lsr-lsr, 12
 lsr-n, 52
 lsr-n-bitp, 52
 lsr-nat-range, 12
 lsr-uint, 9
 lsr-z, 52, 63
 lsr-z-bitp, 52
 lst-int, 119
 lst-integerp, 119
 lst-numberp, 79–81
 lst-numberp0, 79
 lst-of-chrp, 132
 lst-to-bv, 91
 lstcpy, 103, 104
 lstcpy-0, 103
 lstcpy-add1, 104
 lstcpy-cpy1, 104
 lstcpy-lstcpy, 103
 lstcpy-put-nth, 104
 lstcpy1, 104
 lstmcpy, 103, 104
 lstmcpy-cpy, 104
 mapping, 114
 mapping-h, 114
 mbit-means-lessp, 12
 mbit-means-negativep, 66
 mc-ccr, 40
 mc-ccr-range, 40
 mc-ccr-rewrite, 40
 mc-instate, 114
 mc-instate-mem, 114
 mc-mem, 40, 106, 107, 114–118
 mc-mem-rewrite, 40
 mc-pc, 40
 mc-pc-range, 40
 mc-pc-rewrite, 40
 mc-rfile, 40
 mc-rfile-rewrite, 40
 mc-state, 40
 mc-status, 40, 73
 mc-status-rewrite, 40
 mcar, 91, 92, 102
 mcar-mcar, 92
 mcar-nth, 92
 mcd, 91, 92, 102, 103, 126, 130, 134
 mcd-listp-len, 92

mcdm-mcdr, 92
mcdm-nth, 92
mcode-addrp, 74, 79–81, 100, 117
mean-difference-1, 137
mean-difference-2, 137
mean-lessp, 121
mean-lessp-lemma, 121
mem-ilst, 88, 89, 94, 95, 97–99, 101, 107, 119
mem-ilst-int-range, 89
mem-ilst-integerp, 89
mem-ilst-lst-integerp, 119
mem-induct0, 46
mem-induct1, 79
mem-induct2, 93
mem-induct3, 95
mem-induct4, 96
mem-lst, 88, 89, 93–100, 102–104, 106, 119
mem-lst-get-lst, 93
mem-lst-get-lst0, 93
mem-lst-get-vals, 93
mem-lst-get-vals0, 93
mem-lst-integerp, 119
mem-lst-len, 102
mem-lst-lessp, 89
mem-lst-mcar, 102
mem-lst-mcar-1, 102
mem-lst-mcar-2, 102
mem-lst-mcdr, 102
mem-lst-mcdr-0, 102
mem-lst-mcdr-1, 103
mem-lst-mcdr-uint, 102
mem-lst-mcdr-uint-1, 103
mem-lst-nat-range, 89
mem-lst-non-numberp, 89
mem-lst-numberp, 89
mem-lst-plus, 102
mem-lst-put-lst, 95
mem-lst-put-vals, 96
mem-lst-same, 89
memchr, 122
memchr*, 129
memchr-bounds, 134
memchr1, 121, 122, 134
memcmp, 122
memcmp1, 122
memmove, 127
memmove-0, 127, 128
memmove-1, 126, 127
memmove-3, 127, 128
memmove-4, 127, 128
memset, 122
memset1, 122
minus-integerp, 18
misc-group, 116
misc-group-h, 116
mmov1-lst, 105, 126, 127
mmov1-lst-0, 105
mmov1-lst1, 105, 127, 128
mmov1-lst1-0, 105
mmovn-lst, 105, 126
mmovn-lst-0, 105
mmovn-lst1, 105, 127
mmovn-lst1-0, 105
mod-eq, 2
mod32-eq, 2, 45, 81, 82, 106
mod32-eq-deduction0, 106
mod32-eq-deduction1, 106
mod32-eq-deduction2, 106
mod32-eq-deduction3, 106
mode-guards, 114
modn-eq, 43, 44
modn-eq-equal, 44
modn-lst, 101
modn-readm-rn, 101
move-beq-ext, 60
move-beq-int-0, 59
move-beq-int-1, 60
move-beq-uint, 59
move-bgt, 71
move-ble, 72
move-bmi, 68
move-group, 116
move-group-h, 116
move-h, 116
move-ins, 116
move-n, 53, 68, 71, 72

- move-n-bitp, 53
- move-z, 53, 59, 60, 71, 72
- move-z-bitp, 53
- movem-len, 116
- movem-pre-rnlst, 116
- movem-pre-rnlst-len, 116
- movem-rnlst, 116
- movem-rnlst-len, 116
- movem-saved, 118
- movep-write, 115
- movep-write-h, 115
- movep-writep, 115
- movn-lst, 105
- muls, 21, 38
- muls-crock, 21
- muls-n, 51, 69
- muls-n-bitp, 51
- muls-nat-range, 38
- muls-v, 51, 67
- muls-v-bitp, 51
- muls-z, 51, 62, 63
- muls-z-bitp, 51
- muls.1632-beq, 62
- muls.1632-bmi, 69
- muls.1632-bvs, 67
- muls.1632-int, 21
- muls.3232-beq, 62
- muls.3232-bmi, 69
- muls.3232-bvs, 67
- muls.3232-int, 21
- muls.3264-beq, 62
- muls.3264-bmi, 69
- muls.3264-bvs, 67
- muls.3264-int, 21
- mulu, 20, 38
- mulu-n, 51
- mulu-n-bitp, 51
- mulu-nat-range, 38
- mulu-v, 50, 67
- mulu-v-bitp, 50
- mulu-z, 50, 62
- mulu-z-bitp, 50
- mulu.1632-beq, 62
- mulu.1632-bvs, 67
- mulu.1632-nat, 20
- mulu.3232-beq, 62
- mulu.3232-bvs, 67
- mulu.3232-nat, 20
- mulu.3264-beq, 62
- mulu.3264-bvs, 67
- mulu.3264-nat, 20
- n-member, 2, 42, 43
- nat-plus-range, 37
- nat-range, 6–12, 15, 16, 20–30, 34, 35, 37–40, 45, 46, 48, 54, 55, 58–72, 89, 95–99, 135, 138
- nat-range-0, 37
- nat-range-la, 135
- nat-range-of-0, 37
- nat-range-ub, 37
- nat-to-bv, 4, 5
- nat-to-bv-sized, 4–6
- nat-to-bv-sized-head, 5
- nat-to-bv-sized-la0, 5
- nat-to-bv-sized-len, 5
- nat-to-bv-sized-sized-to-nat, 6
- nat-to-bv-sized-to-nat, 6
- nat-to-bv-to-nat, 5
- nat-to-int, 8, 10–13, 16, 21–23, 29, 30, 58–64, 66–72, 77, 78, 80, 86–88, 94, 97–100, 118, 119, 135–137
- nat-to-int-0, 16
- nat-to-int-integerp, 16
- nat-to-int-range, 16
- nat-to-int-to-nat, 16
- nat-to-int=, 16
- nat-to-uint, 9, 15, 16, 20, 22–25, 35, 59–65, 67, 89, 93–98, 101–103, 118, 127, 131, 132, 135–139
- nat-to-uint-range, 16
- nat-to-uint-to-nat, 15
- neg, 2, 25, 27, 30, 38, 53, 54, 88, 106
- neg-add, 54
- neg-cancel, 54

- neg-head, 53
- neg-nat-range, 38
- neg-neg, 53
- negativep-guts0, 18
- not-n, 53
- not-n-bitp, 53
- not-z, 53
- not-z-bitp, 53
- null, 121–124, 128, 133
- numberp-eval\$-add, 55
- numberp-eval\$-add-tree, 55
- numberp-integerp, 18

- or-group, 115
- or-group-h, 115
- or-n, 51
- or-n-bitp, 51
- or-z, 51, 60, 61
- or-z-bitp, 51

- p, 114–116
- pc-byte-read, 36, 38, 45, 47, 74, 79, 90
- pc-byte-read-mcode0, 79
- pc-byte-read-mcode1, 79
- pc-byte-read-mcode2, 79
- pc-byte-read-mcode3, 79
- pc-byte-read-nat-range, 38
- pc-byte-read-write, 45
- pc-byte-read-write-mem, 47
- pc-byte-read=pc-read-mem-1, 90
- pc-byte-readp, 36, 45–47, 74, 75, 100
- pc-byte-readp->byte-readp, 45
- pc-byte-readp-rom0, 75
- pc-byte-readp-rom1, 75
- pc-byte-readp-rom2, 75
- pc-byte-readp-rom3, 75
- pc-read, 36, 44
- pc-read-mem, 36, 46, 47, 80, 90, 100, 117
- pc-read-mem-byte-write, 47
- pc-read-mem-mcode0, 80
- pc-read-mem-mcode1, 80
- pc-read-mem-mcode2, 80
- pc-read-mem-mcode3, 80
- pc-read-mem-nat-range, 46
- pc-read-mem-write-mem, 47
- pc-read-mem-writemem, 100
- pc-read-memp, 2, 36, 47, 74, 75, 81, 100, 116
- pc-read-memp->read-memp, 47
- pc-read-memp-la0, 74
- pc-read-memp-la1, 74
- pc-read-memp-la2, 74
- pc-read-memp-la3, 74
- pc-read-memp-rom0, 75
- pc-read-memp-rom1, 75
- pc-read-memp-rom2, 75
- pc-read-memp-rom3, 75
- pc-read-write, 44
- pc-readp, 35, 43, 44
- pc-readp->readp, 43
- plus-0, 137
- plus-add1-sub1, 27
- plus-difference, 102
- plus-numberp, 24
- plus-times-equal, 44
- plus-times-lessp, 95
- plus-times-sub1, 137
- plus-to-iplus, 13
- plus2-times-sub1, 138
- plus3-times-sub1, 138
- proper-lstp, 91, 92, 96
- put-commutativity, 104
- put-get, 41
- put-get-lst-is-cpy, 104
- put-get-vals-is-cpy, 104
- put-lst, 90, 96, 104
- put-lst-int, 119
- put-lst-of-chrp, 132
- put-nth, 41, 90, 91, 95–99, 103–105, 119, 121–124, 131–133
- put-nth-0, 41
- put-nth-len, 41
- put-put, 41
- put-vals, 91, 93, 96, 104, 105
- put-vals-append, 93

quot, 23, 38
 quot-nat, 23
 quot-nat-range, 38
 quotient-diff, 10
 quotient-diff-la, 10
 quotient-exp-less, 10
 quotient-int-range, 17
 quotient-nat-range, 37
 quotient-plus-times-exp2-1, 31
 quotient-plus-times-exp2-2, 31

 ram-addrp, 2, 73, 74, 77, 78, 99,
 100, 117, 136, 139
 ram-addrp-3, 139
 ram-addrp-4, 139
 ram-addrp-la1, 78
 ram-addrp-la2, 78
 read, 36, 44
 read->pc-read-mem, 80
 read-an, 118
 read-dn, 118
 read-lst, 80, 81
 read-lst0, 79, 80
 read-mem, 36, 46, 48, 80, 81, 88–90,
 93, 94, 98, 101, 106, 117,
 118, 137
 read-mem-byte-write, 48
 read-mem-byte-write-end, 48
 read-mem-ilst, 94
 read-mem-ilst-asl, 98
 read-mem-ilst-int, 94
 read-mem-int-16_32, 137
 read-mem-int-8_16, 137
 read-mem-int-8_32, 137
 read-mem-lst, 94
 read-mem-lst-asl, 98
 read-mem-lst-int, 94
 read-mem-lst-la, 93
 read-mem-lst0, 94
 read-mem-mcode1-int, 80
 read-mem-mcode2, 81
 read-mem-mcode3, 81
 read-mem-nat-range, 46
 read-mem-non-numberp, 89

 read-mem-plus, 106
 read-memp, 36, 47, 76–78, 99, 100,
 116, 136
 read-memp-ram0, 78
 read-memp-ram1, 78
 read-memp-ram1-asl, 99
 read-memp-ram1-int, 78
 read-memp-ram1-uint, 136
 read-memp-ram2, 78
 read-memp-ram3, 78
 read-memp-rom0, 76
 read-memp-rom1, 76
 read-memp-rom1-asl, 99
 read-memp-rom1-int, 77
 read-memp-rom2, 76
 read-memp-rom3, 76
 read-rn, 38, 41–43, 120, 136, 137
 read-rn-0, 43
 read-rn-equal*, 120
 read-rn-int-16_32, 137
 read-rn-int-8_16, 137
 read-rn-int-8_32, 136
 read-rn-nat-range, 38
 read-write, 44
 read-write-mem-end, 48
 read-write-mem1, 48
 read-write-mem2, 48
 read-write-rn, 41
 read-write-rn-end, 41
 read-writem-mem, 101
 read-writem-rn, 42
 read-writem-rn-end, 42, 43
 readm-mem, 37, 89, 101, 106, 107,
 118
 readm-mem-lst, 89
 readm-rn, 42, 43, 101
 readm-rn-len, 42
 readm-write-mem, 101
 readm-write-rn, 43
 readm-writem-mem, 101
 readp, 35, 43
 rem, 22, 38
 rem-nat, 22
 rem-nat-range, 38

- remainder-diff, 10
- remainder-diff-la, 9
- remainder-int-range, 17
- remainder-leq, 26
- remainder-plus-times-exp2-1, 31
- remainder-plus-times-exp2-2, 31
- remainder-quotient-exp2, 31
- remainder2-plus-times-exp2, 31
- replace, 9, 32–34, 41
- replace-0, 9
- replace-associativity, 32
- replace-head, 32
- replace-leq, 32
- replace-leq1, 32
- replace-nat-range, 34
- replace-reflex, 32
- rn-saved, 118
- rol-c, 51
- rol-c-bitp, 51
- rol-n, 51
- rol-n-bitp, 51
- rol-z, 51
- rol-z-bitp, 51
- rom-addrp, 2, 73, 75–77, 99, 100, 117
- rom-addrp-la1, 76
- rom-addrp-la2, 76
- ror-c, 51
- ror-c-bitp, 51
- ror-n, 51
- ror-n-bitp, 51
- ror-z, 51
- ror-z-bitp, 51
- roxl-c, 52
- roxl-c-bitp, 52
- roxl-n, 52
- roxl-n-bitp, 52
- roxl-z, 52
- roxl-z-bitp, 52
- roxr-c, 53
- roxr-c-bitp, 53
- roxr-n, 53
- roxr-n-bitp, 53
- roxr-z, 53
- roxr-z-bitp, 53
- rts-addr, 118
- s&r-group, 116
- s&r-group-h, 116
- scc-group, 115
- scc-group-h, 115
- set-cvznx, 39
- set-cvznx-c, 39
- set-cvznx-ccr, 39
- set-cvznx-n, 39
- set-cvznx-nat-range, 39
- set-cvznx-v, 39
- set-cvznx-x, 39
- set-cvznx-z, 39
- set-set-cvznx1, 39
- set-set-cvznx2, 39
- slen, 133, 134
- slen-01, 133
- slen-add1, 133
- slen-lbound, 133
- slen-put, 133
- slen-put0, 133
- slen-rec, 134
- slen-ubound, 133
- splus, 73
- stepi, 73, 116
- stepi-h, 116
- stepi-p, 116
- stepn, 73, 106, 107, 116, 117
- stepn-h, 116
- stepn-lemma, 73
- stepn-mcode-addrp, 117
- stepn-mem-ilst, 107
- stepn-mem-lst, 106
- stepn-pc-read-mem, 117
- stepn-pc-read-memp, 116
- stepn-ram-addrp, 117
- stepn-read-mem, 117
- stepn-read-memp, 116
- stepn-readm-mem, 106
- stepn-rewriter, 73
- stepn-rewriter0, 73
- stepn-rom-addrp, 117

stepn-write-memp, 117
 strcat, 123
 strchr, 125, 129, 134
 strchr*, 129
 strchr-bounds, 134
 strchr1, 125, 126, 129, 130
 strchr1*, 129, 130
 strchr1-bounds, 125
 strchr1-false-0, 125
 strcmp, 123
 strcoll, 123
 strcpy, 122
 strcpy1, 123
 strcpy2, 123, 124
 strcspn, 125, 131, 135
 strcspn*, 129, 131
 strcspn-bounds, 135
 stringp, 133, 134
 stringp-la, 134
 strlen, 122–124, 126, 130
 strlen*, 128
 strncat, 124
 strncmp, 124, 126, 130
 strncmp1, 124
 strncpy, 124
 strncpy1, 124
 strpbrk, 126, 134
 strpbrk*, 129
 strpbrk-bounds, 134
 strrchr, 125, 134
 strrchr*, 129
 strrchr-bounds, 134
 strspn, 126, 130–132, 134, 135
 strspn*, 130, 132
 strspn-bounds, 135
 strspn-ubound, 134
 strstr, 126
 strstr*, 130
 strstr-bounds, 134
 strstr1, 126, 134
 strstr1*, 130
 strtok-last, 132
 strtok-last0, 131
 strtok-last1, 131, 132
 strtok-lst, 131
 strtok-lst0, 131
 strtok-lst1, 131
 strtok-lst2, 131
 strtok-tok, 130
 strxfrm, 132
 strxfrm1, 132
 sub, 2, 23, 26–29, 54, 56, 75, 118
 sub-0, 26
 sub-add, 28
 sub-adder, 27
 sub-bcs&cc, 65
 sub-beq-ext, 60
 sub-beq-int-0, 60
 sub-beq-int-1, 60
 sub-beq-uint, 60
 sub-bge, 70
 sub-bgt, 71
 sub-bhi-0, 59
 sub-bhi-1, 59
 sub-bhi-int, 58
 sub-ble, 71
 sub-bls, 59
 sub-bltn, 70
 sub-bmi, 68
 sub-bvs&vc, 67
 sub-c, 50, 58, 59, 65
 sub-c-bitp, 50
 sub-cancel, 28
 sub-cancel0, 28
 sub-equal-0, 27
 sub-group, 115
 sub-group-h, 115
 sub-int, 30
 sub-leq-1, 54
 sub-leq-2, 54
 sub-leq-la, 26
 sub-n, 50, 68, 70, 71
 sub-n-bitp, 50
 sub-nat-la, 23
 sub-nat-range, 27
 sub-neg, 54
 sub-sub, 28
 sub-sub1, 28

- sub-subx-c, 58
- sub-subx-v, 66
- sub-uint, 25
- sub-v, 50, 66, 67, 70, 71
- sub-v-bitp, 50
- sub-x-x, 26
- sub-z, 50, 58–60, 71
- sub-z-bitp, 50
- sub-z-la, 58
- sub-z-la1, 71
- sub1-int-range, 17
- sub1-nat-range, 37
- sub1-times2-nat-range, 37
- subbagp, 55
- subtractor, 24, 26, 30
- subtractor-int, 30
- subtractor-nat-la, 24
- subtractor-nat-range, 26
- subtractor-uint, 24
- subx-addx-v, 66
- subx-c, 50, 58
- subx-c-bitp, 50
- subx-c-la, 58
- subx-n, 50
- subx-n-bitp, 50
- subx-v, 50, 66
- subx-v-bitp, 50
- subx-v-la, 66
- subx-z, 50
- subx-z-bitp, 50
- swap, 121
- swap-n, 53
- swap-n-bitp, 53
- swap-z, 53
- swap-z-bitp, 53
- t3, 114, 116, 117
- tail, 8, 9, 31–34, 37, 49, 65, 90, 91, 95, 96
- tail-0, 8
- tail-app, 32
- tail-equal-0, 9
- tail-head, 32
- tail-lemma, 9
- tail-leq, 8
- tail-lessp, 8
- tail-lst, 79, 80
- tail-mbit, 65
- tail-nat-range, 37
- tail-of-0, 8
- tail-replace, 32
- tail-tail, 31
- times-exp2-nat-range, 37
- times-lessp-cancel1, 99
- times-lessp_1, 20
- times-lessp_2, 21
- times-lessp_3, 21
- times-lessp_4, 21
- times-plus-lessp-cancel, 86
- times-sub1, 11
- uint-range, 9, 15, 16, 20, 23, 48, 62, 63, 65, 95–99, 101
- uint-range-la, 15
- uint-to-nat, 15
- uint-to-nat-range, 15
- uint-to-nat-to-uint, 15
- uread-an, 118
- uread-dn, 118
- uread-mem, 118
- vec, 3, 4, 6
- write, 36, 44
- write-else-mem-ilst, 95
- write-else-mem-lst, 94
- write-mem, 36, 46–49, 73, 74, 81, 90, 94–99, 101, 114, 115
- write-mem-byte-write, 49
- write-mem-ilst, 97
- write-mem-ilst-asl, 99
- write-mem-ilst-int, 98
- write-mem-ilst0, 97
- write-mem-lst, 96
- write-mem-lst-asl, 98
- write-mem-lst-int, 97
- write-mem-lst-la, 95
- write-mem-lst0, 97

write-mem-maintain-byte-readp, 46
 write-mem-maintain-byte-writep, 47
 write-mem-maintain-movep-writep, 115
 write-mem-maintain-pc-byte-readp, 46
 write-mem-maintain-pc-read-memp, 47
 write-mem-maintain-ram-addrp, 74
 write-mem-maintain-read-memp, 47
 write-mem-maintain-rom-addrp, 73
 write-mem-maintain-write-memp, 47
 write-mem-mcode-addrp, 81
 write-memp, 2, 36, 47, 74, 75, 77, 81, 90, 100, 114, 116, 117, 136
 write-memp->read-memp, 47
 write-memp-la0, 74
 write-memp-la1, 74
 write-memp-la2, 74
 write-memp-la3, 75
 write-memp-ram0, 77
 write-memp-ram1, 77
 write-memp-ram1-asl, 99
 write-memp-ram1-int, 77
 write-memp-ram1-uint, 136
 write-memp-ram2, 77
 write-memp-ram3, 77
 write-rn, 41–43
 write-rn-len, 42
 write-write, 44
 write-write-induct, 49
 write-write-la, 44
 write-write-mem, 49
 write-write-rn, 42
 writem-else-mem-ilst, 101
 writem-else-mem-lst, 100
 writem-mem, 100, 101, 116
 writem-mem-h, 116
 writem-mem-maintain-byte-readp, 100
 writem-mem-maintain-byte-writep, 100
 writem-mem-maintain-pc-byte-readp, 100
 writem-mem-maintain-ram-addrp, 100
 writem-mem-maintain-read-memp, 100
 writem-mem-maintain-rom-addrp, 100
 writem-mem-maintain-write-memp, 100
 writem-mem-mcode-addrp, 100
 writem-rn, 42
 writep, 36, 43, 44
 writep->readp, 43
 z-flag-la, 63
 zero-list, 124
 zero-list1, 124