```
#|

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

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

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

```
; x is an element of lst wrt numberp.
```

DEFINITION:
n-member $(x,\ lst)$
$=$ **if** $lst \simeq$ **nil then f**
else (fix (x) = fix (car (lst))) $\vee$ n-member $(x$, 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:
mod-eq $(n,\ x,\ y)$
$=$ **if** $n \simeq 0$ **then t**
else (bcar (x) = bcar (y)) $\wedge$ mod-eq $(n - 1$, bcdr (x), bcdr $(y))$ **endif**

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

DEFINITION: mod32-eq $(x,\ y)$ = (head $(x,\ 32)$ = 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: neg $(n,\ x)$ = sub $(n,\ x,\ 0)$

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

DEFINITION:
disjoint0 $(x,\ m,\ y)$
$=$ **if** $m \simeq 0$ **then t**
else ($\neg$ mod32-eq (add $(32,\ x,\ m - 1)$, y))
$\wedge$ 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 $(x,\ m,\ \text{add}\,(32,\ y,\ n-1)) \wedge \text{disjoint}\,(x,\ m,\ y,\ n-1)$ **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** $bv \simeq$ **nil then** B0
elseif b0p (car (bv)) **then** B0
else B1 **endif**

DEFINITION:
vec (bv)
$=$ **if** $bv \simeq$ **nil then nil**
else cdr (bv) **endif**

DEFINITION:
fix-bv (bv)
$=$ **if** $bv \simeq$ **nil then nil**
else cons (bit (bv), fix-bv (vec (bv))) **endif**

DEFINITION:
bv-len (bv)
$=$ **if** $bv \simeq$ **nil then** 0
else $1 +$ bv-len (vec (bv)) **endif**

DEFINITION:
bv-not (bv)
$=$ **if** $bv \simeq$ **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
listp (bv-head $(x,\ n)) = (n \not\simeq \mathtt{0})$

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

```
;                  BV-LEN
```

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

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

THEOREM: bv-adder-len
bv-len (bv-adder $(c,\ x,\ y))$ = 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
nat-to-bv-sized (n **mod** exp $(\mathtt{2},\ k),\ k)$ = nat-to-bv-sized $(n,\ k)$

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

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

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

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

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

THEOREM: nat-to-bv-to-nat
bv-to-nat (nat-to-bv $(n))$ = fix (n)

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

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

THEOREM: nat-to-bv-sized-to-nat
bv-to-nat (nat-to-bv-sized (m, n))
= **if** $m < \exp(2, n)$ **then** fix (m)
else m **mod** $\exp(2, n)$ **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** $m < \exp(2, n)$ **then** fix (m)
else m **mod** $\exp(2, n)$ **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** $i <$ bv-len (x) **then** b-not (bv-bitn (x, i))
else B0 **endif**

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

```
;             BV-ADDER
; a bridge function.
```

DEFINITION:
bv-adder&carry (c, x, y)
= **if** $x \simeq$ **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
bv-adder $(c,\ x,\ y) =$ bv-head (bv-adder&carry $(c,\ x,\ y)$, bv-len (x))

```
;                    lifting lemmas
```

THEOREM: bv-not-lognot
bv-to-nat (bv-not (x)) $=$ lognot (bv-len (x), bv-to-nat (x))

THEOREM: bv-bitn-bitn
bv-bitn $(x,\ n) =$ bitn (bv-to-nat (x), n)

THEOREM: bv-mbit-bitn
bv-mbit (x)
$=$ **if** bv-len $(x) = 0$ **then** 0
else bitn (bv-to-nat (x), bv-len $(x) - 1$) **endif**

THEOREM: bv-adder&carry-nat
(bv-len $(x) =$ bv-len (y))
$\rightarrow$ (bv-to-nat (bv-adder&carry $(c,\ x,\ y)$)
$=$ (fix-bit (c) $+$ bv-to-nat (x) $+$ bv-to-nat (y)))

THEOREM: bv-adder-nat
((bv-len $(x) =$ bv-len (y)) $\wedge$ bitp (c))
$\rightarrow$ (bv-to-nat (bv-adder $(c,\ x,\ y)$)
$=$ adder (bv-len (x), c, bv-to-nat (x), bv-to-nat (y)))

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

THEOREM: add-evenp
(evenp (x) $\wedge$ evenp (y)) $\rightarrow$ evenp (add $(n,\ x,\ y)$)

EVENT: Disable evenp.

```
;                         LOGNOT
```

THEOREM: lognot-0
$(x \notin \mathbf{N})$
$\rightarrow$ ((lognot $(x,\ y) =$ lognot $(0,\ y)$) $\wedge$ (lognot $(y,\ x) =$ lognot $(y,\ 0)$))

THEOREM: lognot-nat-rangep
nat-rangep (lognot $(n,\ x)$, n)

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

THEOREM: lognot-cancel
$(\text{nat-rangep}\,(x,\ n) \land \text{nat-rangep}\,(y,\ n))$
$\to \quad ((\text{lognot}\,(n,\ x) = \text{lognot}\,(n,\ y)) = (\text{fix}\,(x) = \text{fix}\,(y)))$

```
; the integer interpretation of lognot.
```

THEOREM: lognot-int
$(\text{nat-rangep}\,(x,\ n) \land (n \not\simeq \mathtt{0}))$
$\to \quad (\text{nat-to-int}\,(\text{lognot}\,(n,\ x),\ n) = \text{iplus}\,(\mathtt{-1},\ \text{ineg}\,(\text{nat-to-int}\,(x,\ n))))$

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

EVENT: Disable lognot.

```
;                BASICS ABOUT HEAD & TAIL
```

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

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

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

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

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

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

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

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

THEOREM: tail-equal-0
$(\text{tail}\,(x,\ n) = \mathtt{0}) = \text{nat-rangep}\,(x,\ n)$

THEOREM: head-lemma
$\text{nat-rangep}\,(x,\ n) \rightarrow (\text{head}\,(x,\ n) = \text{fix}\,(x))$

THEOREM: tail-lemma
$\text{nat-rangep}\,(x,\ n) \rightarrow (\text{tail}\,(x,\ n) = \mathtt{0})$

THEOREM: replace-0
$(\text{replace}\,(\mathtt{0},\ x,\ y) = \text{fix}\,(y))$
$\wedge\quad (\text{replace}\,(n,\ x,\ \mathtt{0}) = \text{head}\,(x,\ n))$
$\wedge\quad ((\text{nat-rangep}\,(x,\ n) \wedge \text{nat-rangep}\,(y,\ n))$
$\qquad \rightarrow\quad (\text{replace}\,(n,\ x,\ y) = \text{fix}\,(x)))$

THEOREM: app-0
$(\text{app}\,(n,\ x,\ \mathtt{0}) = \text{head}\,(x,\ n))$
$\wedge\quad (\text{app}\,(n,\ \mathtt{0},\ y) = (y * \exp\,(\mathtt{2},\ n)))$
$\wedge\quad (\text{app}\,(\mathtt{0},\ x,\ y) = \text{fix}\,(y))$

```
;                 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
$\text{uint-rangep}\,(x,\ n - \mathit{cnt})$
$\rightarrow\quad (\text{nat-to-uint}\,(\text{lsl}\,(n,\ x,\ \mathit{cnt})) = (\text{nat-to-uint}\,(x) * \exp\,(\mathtt{2},\ \mathit{cnt})))$

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

```
; two lemmas for asl-int.
```

THEOREM: remainder-diff-la
$(((x \ \mathbf{mod}\ z) = \mathtt{0}) \wedge (y < z))$
$\rightarrow\quad (((x - y)\ \mathbf{mod}\ z)$
$\qquad = \quad \mathbf{if}\ x \le y\ \mathbf{then}\ \mathtt{0}$
$\qquad\qquad \mathbf{elseif}\ y \simeq \mathtt{0}\ \mathbf{then}\ \mathtt{0}$
$\qquad\qquad \mathbf{else}\ z - y\ \mathbf{endif})$

THEOREM: asl-int-crock1
$(x < \exp\,(\mathtt{2},\ (n - s) - 1))$
$\rightarrow\quad (\text{head}\,(x * \exp\,(\mathtt{2},\ s),\ n) = (x * \exp\,(\mathtt{2},\ s)))$

THEOREM: asl-int-crock2
$(((\exp(2, s) * (\exp(2, n) - x)) \le \exp(2, n - 1)) \land (x < \exp(2, n)))$
$\rightarrow \quad (\text{head}(x * \exp(2, s), n)$
$\quad = \quad (\exp(2, n) - (\exp(2, s) * (\exp(2, n) - x))))$

THEOREM: asl-int
$(\text{nat-rangep}(x, n) \land \text{int-rangep}(\text{nat-to-int}(x, n), n - s))$
$\rightarrow \quad (\text{nat-to-int}(\text{asl}(n, x, s), n) = \text{itimes}(\text{nat-to-int}(x, n), \exp(2, s)))$

EVENT: Disable asl-int-crock1.

EVENT: Disable asl-int-crock2.

THEOREM: quotient-diff-la
$(((x \textbf{ mod } z) = 0) \land (y < z))$
$\rightarrow \quad (((x - y) \div z)$
$\quad = \quad \textbf{if } y \simeq 0 \textbf{ then } x \div z$
$\quad\quad \textbf{else } (x \div z) - 1 \textbf{ endif})$

THEOREM: remainder-diff
$((x \textbf{ mod } z) = 0)$
$\rightarrow \quad (((x - y) \textbf{ mod } z)$
$\quad = \quad \textbf{if } (x \le y) \lor ((y \textbf{ mod } z) = 0) \textbf{ then } 0$
$\quad\quad \textbf{else } z - (y \textbf{ mod } z) \textbf{ endif})$

THEOREM: quotient-diff
$((x \textbf{ mod } z) = 0)$
$\rightarrow \quad (((x - y) \div z)$
$\quad = \quad \textbf{if } (y \textbf{ mod } z) = 0 \textbf{ then } (x \div z) - (y \div z)$
$\quad\quad \textbf{else } ((x \div z) - (y \div z)) - 1 \textbf{ endif})$

THEOREM: quotient-exp-lessp
$(cnt \le n)$
$\rightarrow \quad (((x \div \exp(2, cnt)) < \exp(2, n - cnt)) = (x < \exp(2, n)))$

THEOREM: lessp-times-exp-1s
$((m + n) = \text{fix}(k))$
$\rightarrow \quad (((x + (\exp(2, m) * (\exp(2, n) - 1))) < \exp(2, k - 1))$
$\quad = \quad \textbf{if } n \simeq 0 \textbf{ then } x < \exp(2, (m + n) - 1)$
$\quad\quad \textbf{else f endif})$

THEOREM: lessp-app-1s
$((m + n) = \text{fix}(k))$
$\rightarrow \quad ((\text{app}(m, x, \exp(2, n) - 1) < \exp(2, k - 1))$
$\quad = \quad \textbf{if } n \simeq 0 \textbf{ then } \text{head}(x, m) < \exp(2, (m + n) - 1)$
$\quad\quad \textbf{else f endif})$

THEOREM: times-sub1
$((y - 1) * x) = ((x * y) - x)$

THEOREM: difference-app-1s
$((m + n) = \text{fix}(k))$
$\rightarrow \quad ((\exp(2, k) - \text{app}(m, x, \exp(2, n) - 1)) = (\exp(2, m) - \text{head}(x, m)))$

EVENT: Disable times-sub1.

THEOREM: asr-int-crock
$((x < \exp(2, n)) \wedge (x \not< \exp(2, n - 1)) \wedge (cnt \leq n))$
$\rightarrow \quad ((\text{app}(n - cnt, x \div \exp(2, cnt), \exp(2, cnt) - 1) < \exp(2, n - 1))$
$\quad\quad = \quad \mathbf{f})$

THEOREM: asr-int
nat-rangep (x, n)
$\rightarrow$ (nat-to-int (asr (n, x, i), n)
= **if** negativep (nat-to-int (x, n))
then if iremainder (nat-to-int (x, n), exp $(2, i)$) = 0
then iquotient (nat-to-int (x, n), exp $(2, i)$)
else iplus (-1, iquotient (nat-to-int (x, n), exp $(2, i)$)) **endif**
else iquotient (nat-to-int (x, n), exp $(2, i)$) **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-rangep
nat-rangep (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-rangep
nat-rangep $(x,\ n) \rightarrow$ nat-rangep (lsr $(x,\ cnt),\ n)$

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

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

THEOREM: asl-nat-rangep
nat-rangep (asl $(n,\ x,\ cnt),\ n)$

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

THEOREM: asr-0
nat-rangep $(x,\ n) \rightarrow$ (asr $(n,\ x,\ 0) =$ fix $(x))$

THEOREM: asr-nat-rangep
nat-rangep $(x,\ n) \rightarrow$ nat-rangep (asr $(n,\ x,\ cnt),\ n)$

```
; unproved!  Not useful yet.
; (prove-lemma asr-asr (rewrite)
;     (implies (and (nat-rangep 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
nat-rangep $(x,\ n)$
$\rightarrow$ (bitn $(x,\ n-1)$
= **if** $x < \exp(2,\ n-1)$ **then** 0
else 1 **endif**)

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

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

THEOREM: ext-int
(nat-rangep $(x,\ n) \wedge (n < size))$
$\rightarrow$ (nat-to-int (ext $(n,\ x,\ size),\ size) =$ 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) $\wedge$ integerp (y) $\wedge$ int-rangep $(x,\ n)$ $\wedge$ int-rangep $(y,\ n)$)
$\rightarrow$ (adder-int-end $(n,\ c,\ x,\ y)$
$=$ **if** int-rangep (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) $\wedge$ integerp (y) $\wedge$ bitp (z) $\wedge$ negativep (x) $\wedge$ negativep (y))
$\rightarrow$ negativep (iplus $(x,$ iplus $(y,\ z)))$

THEOREM: iplus-with-carry-non-negativep
(integerp (x)
$\wedge$ integerp (y)
$\wedge$ bitp (z)
$\wedge$ ($\neg$ negativep (x))
$\wedge$ ($\neg$ negativep (y)))
$\rightarrow$ ($\neg$ 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)
$\wedge$ integerp (y)
$\wedge$ bitp (z)
$\wedge$ int-rangep $(x,\ n)$
$\wedge$ int-rangep $(y,\ n)$)
$\rightarrow$ ($\neg$ negativep (iplus $(x,$ iplus $(y,$ iplus $(z,$ exp $(2,\ n))))$))

THEOREM: addx-v-crock2
$$\begin{aligned}
&(\text{integerp}\,(x) \\
&\ \wedge\ \ \text{integerp}\,(y) \\
&\ \wedge\ \ \text{bitp}\,(z) \\
&\ \wedge\ \ (n \not\simeq 0) \\
&\ \wedge\ \ \text{int-rangep}\,(x,\ n) \\
&\ \wedge\ \ \text{int-rangep}\,(y,\ n)) \\
&\rightarrow\ \ \text{negativep}\,(\text{iplus}\,(x,\ \text{iplus}\,(y,\ \text{iplus}\,(z,\ -\ \exp\,(2,\ n)))))
\end{aligned}$$

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

THEOREM: add-bmi-crock1
$$\begin{aligned}
&(\text{integerp}\,(x) \wedge \text{integerp}\,(y) \wedge \text{int-rangep}\,(x,\ n) \wedge \text{int-rangep}\,(y,\ n)) \\
&\rightarrow\ \ (\neg\ \text{negativep}\,(\text{iplus}\,(x,\ \text{iplus}\,(y,\ \exp\,(2,\ n)))))
\end{aligned}$$

THEOREM: add-bmi-crock2
$$\begin{aligned}
&(\text{integerp}\,(x) \wedge \text{integerp}\,(y) \wedge \text{int-rangep}\,(x,\ n) \wedge \text{int-rangep}\,(y,\ n)) \\
&\rightarrow\ \ \text{negativep}\,(\text{iplus}\,(x,\ \text{iplus}\,(y,\ -\ \exp\,(2,\ n))))
\end{aligned}$$

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

THEOREM: ilessp-crock1
$$\begin{aligned}
&(\text{integerp}\,(x) \wedge \text{integerp}\,(y) \wedge \text{int-rangep}\,(x,\ n) \wedge \text{int-rangep}\,(y,\ n)) \\
&\rightarrow\ \ (\neg\ \text{ilessp}\,(\text{iplus}\,(y,\ \exp\,(2,\ n)),\ x))
\end{aligned}$$

THEOREM: ilessp-crock2
$$\begin{aligned}
&(\text{integerp}\,(x) \wedge \text{integerp}\,(y) \wedge \text{int-rangep}\,(x,\ n) \wedge \text{int-rangep}\,(y,\ n)) \\
&\rightarrow\ \ \text{ilessp}\,(\text{iplus}\,(y,\ -\ \exp\,(2,\ n)),\ x)
\end{aligned}$$

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

THEOREM: iplus-int-rangep1
$$\begin{aligned}
&(\text{integerp}\,(x) \\
&\ \wedge\ \ \text{integerp}\,(y) \\
&\ \wedge\ \ \text{bitp}\,(z) \\
&\ \wedge\ \ \text{int-rangep}\,(x,\ n) \\
&\ \wedge\ \ \text{int-rangep}\,(y,\ n) \\
&\ \wedge\ \ (\neg\ \text{negativep}\,(x)) \\
&\ \wedge\ \ \text{negativep}\,(y)) \\
&\rightarrow\ \ \text{int-rangep}\,(\text{iplus}\,(x,\ \text{iplus}\,(y,\ z)),\ n)
\end{aligned}$$

THEOREM: iplus-int-rangep2
(integerp (x)
$\wedge$ integerp (y)
$\wedge$ bitp (z)
$\wedge$ int-rangep $(x,\ n)$
$\wedge$ int-rangep $(y,\ n)$
$\wedge$ negativep (x)
$\wedge$ ($\neg$ negativep (y)))
$\rightarrow$ int-rangep (iplus $(x,$ iplus $(y,\ z)),\ n)$

THEOREM: idifference-int-rangep1
(int-rangep $(x,\ n)$
$\wedge$ int-rangep $(y,\ n)$
$\wedge$ ($\neg$ ilessp $(x,\ y)$)
$\wedge$ ($\neg$ negativep (y)))
$\rightarrow$ int-rangep (idifference $(y,\ x),\ n)$

THEOREM: idifference-int-rangep2
(integerp (x)
$\wedge$ integerp (y)
$\wedge$ int-rangep $(x,\ n)$
$\wedge$ int-rangep $(y,\ n)$
$\wedge$ ($\neg$ ilessp $(y,\ x)$)
$\wedge$ negativep (y))
$\rightarrow$ int-rangep (idifference $(y,\ x),\ n)$

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

THEOREM: uint-rangep-la
$(x <$ exp $(2,\ n)) \rightarrow$ uint-rangep $(x,\ n)$

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

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

THEOREM: nat-to-uint-to-nat
uint-to-nat (nat-to-uint $(x)) =$ fix (x)

THEOREM: uint-to-nat-to-uint
nat-to-uint (uint-to-nat $(x)) =$ fix (x)

THEOREM: uint-to-nat-rangep
$(x <$ exp $(2,\ n)) \rightarrow$ nat-rangep (uint-to-nat $(x),\ n)$

THEOREM: nat-to-uint-rangep
nat-rangep $(x,\ n) \rightarrow$ uint-rangep (nat-to-uint $(x),\ n)$

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

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

THEOREM: int-to-nat-0
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-integerp
integerp (nat-to-int $(x,\ n)) =$ nat-rangep $(x,\ n)$

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

THEOREM: nat-to-int-rangep
int-rangep (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-rangep
(integerp $(x) \land$ int-rangep $(x,\ n)) \rightarrow$ nat-rangep (int-to-nat $(x,\ n),\ n)$

THEOREM: nat-to-int-to-nat
nat-rangep $(x,\ n) \rightarrow$ (int-to-nat (nat-to-int $(x,\ n),\ n) =$ fix $(x))$

THEOREM: int-to-nat-to-int
(integerp $(x) \land$ int-rangep $(x,\ n))$
$\rightarrow$ (nat-to-int (int-to-nat $(x,\ n),\ n) =$ fix-int $(x))$

THEOREM: int-to-nat=0
(int-rangep $(x,\ n) \land$ integerp $(x))$
$\rightarrow$ ((int-to-nat $(x,\ n) = 0) = (x = 0))$

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

THEOREM: nat-to-int=
(nat-rangep $(x,\ n) \land$ nat-rangep $(y,\ n))$
$\rightarrow$ ((nat-to-int $(x,\ n) =$ nat-to-int $(y,\ n)) =$ (fix $(x) =$ 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
$(\text{int-rangep}\,(x,\ n) \land (\text{abs}\,(y) < \text{abs}\,(x))) \rightarrow \text{int-rangep}\,(y,\ n)$

```
; 0 is always in "good" integer range.
```

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

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
$(\text{int-rangep}\,(y,\ n) \land (y \not\simeq \mathtt{0})) \rightarrow \text{int-rangep}\,(x \ \mathbf{mod}\ y,\ n)$

```
; iquotient is almost always in the integer range.
```

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

```
; iremainder is almost always in the integer range.
```

THEOREM: iremainder-int-rangep
$$\begin{array}{l} (\text{int-rangep}\,(y,\ n) \land \text{integerp}\,(y) \land (y \neq \mathtt{0})) \\ \rightarrow \quad \text{int-rangep}\,(\text{iremainder}\,(x,\ y),\ n) \end{array}$$

```
; lemmas about integerp.
```

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

THEOREM: fix-int-integerp
integerp (fix-int (x))

THEOREM: minus-integerp
integerp ($-\ x$) $=$ ($x \not\simeq \mathtt{0}$)

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

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

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

THEOREM: iplus-integerp
integerp (iplus (x, y))

THEOREM: ineg-integerp
integerp (ineg (x))

THEOREM: idifference-integerp
integerp (idifference (x, y))

THEOREM: itimes-integerp
integerp (itimes (x, y))

THEOREM: iremainder-integerp
integerp (iremainder (x, y))

THEOREM: iquotient-integerp
integerp (iquotient (x, y))

```
; theorems about iplus, idifference, and ineg.
```

THEOREM: iplus-commutativity
iplus (x, y) $=$ iplus (y, x)

THEOREM: iplus-commutativity1
iplus (x, iplus (y, z)) $=$ iplus (y, iplus (x, z))

THEOREM: iplus-associativity
iplus (iplus (x, y), z) $=$ iplus (x, iplus (y, z))

THEOREM: ineg-iplus
ineg (iplus (x, y)) $=$ iplus (ineg (x), ineg (y))

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

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

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

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

```
; lemmas about ilessp.
```

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

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

```
; lemmas about itimes.
```

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

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

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
$(\neg\ \text{izerop}(x))$
$\rightarrow \quad ((\text{itimes}(x, y) = \text{itimes}(x, z)) = (\text{fix-int}(y) = \text{fix-int}(z)))$

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

```
; lemmas about iremainder and iquotient.
```

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

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

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

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

EVENT: Disable iplus.

EVENT: Disable itimes.

EVENT: Disable iremainder.

EVENT: Disable iquotient.

EVENT: Disable integerp.

```
; unsigned interpretation of mulu.
```

THEOREM: times-lessp_1
$((x < \mathit{x1}) \land (y < \mathit{y1})) \rightarrow ((x * y) < (\mathit{x1} * \mathit{y1}))$

```
; three instances of mulu-nat.
```

THEOREM: mulu_1632-nat
$(\text{nat-rangep}\,(x,\ \mathtt{16}) \land \text{nat-rangep}\,(y,\ \mathtt{16}))$
$\rightarrow\quad (\text{nat-to-uint}\,(\text{mulu}\,(\mathtt{32},\ x,\ y,\ \mathtt{16}))$
$\qquad = \quad (\text{nat-to-uint}\,(x) * \text{nat-to-uint}\,(y)))$

THEOREM: mulu_3264-nat
$(\text{nat-rangep}\,(x,\ \mathtt{32}) \land \text{nat-rangep}\,(y,\ \mathtt{32}))$
$\rightarrow\quad (\text{nat-to-uint}\,(\text{mulu}\,(\mathtt{64},\ x,\ y,\ \mathtt{32}))$
$\qquad = \quad (\text{nat-to-uint}\,(x) * \text{nat-to-uint}\,(y)))$

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

```
; signed interpretation of muls.
```

THEOREM: exp2-lessp-crock
$((i \not\simeq 0)$
$\land$ **if** $k < i$ **then f**
else t endif)
$\rightarrow$ $(\exp(2, i - 1) < \exp(2, k))$

THEOREM: head-int-crock
(integerp $(x) \land$ int-rangep $(x, n) \land (n \leq j))$
$\rightarrow$ (head (int-to-nat $(x, j), n)$ = int-to-nat (x, n))

EVENT: Disable exp2-lessp-crock.

THEOREM: times-lessp_2
$((x \leq x1) \land (y \leq y1) \land (x1 \not\simeq 0) \land (y1 \not\simeq 0))$
$\rightarrow$ $((x * y) < (2 * x1 * y1))$

THEOREM: times-lessp_3
$((x \leq \exp(2, n - 1)) \land (y \leq \exp(2, n - 1)) \land (n \not\simeq 0))$
$\rightarrow$ $(((x * y) < \exp(2, (n + n) - 1)) = \mathbf{t})$

THEOREM: times-lessp_4
$((x \leq \exp(2, n - 1)) \land (y \leq \exp(2, n - 1)) \land (n \not\simeq 0))$
$\rightarrow$ $((\exp(2, (n + n) - 1) < (x * y)) = \mathbf{f})$

THEOREM: muls-crock
(int-rangep $(x, n) \land$ int-rangep $(y, n) \land (n \not\simeq 0))$
$\rightarrow$ int-rangep (itimes $(x, y), n + n)$

```
; three instances of muls-int.
```

THEOREM: muls_1632-int
(nat-rangep $(x, 16) \land$ nat-rangep $(y, 16))$
$\rightarrow$ (nat-to-int (muls $(32, x, y, 16), 32)$
= itimes (nat-to-int $(x, 16)$, nat-to-int $(y, 16)$))

THEOREM: muls_3264-int
(nat-rangep $(x, 32) \land$ nat-rangep $(y, 32))$
$\rightarrow$ (nat-to-int (muls $(64, x, y, 32), 64)$
= itimes (nat-to-int $(x, 32)$, nat-to-int $(y, 32)$))

THEOREM: muls_3232-int
int-rangep (itimes (nat-to-int $(x, 32)$, nat-to-int $(y, 32)$), 32)
$\rightarrow$ (nat-to-int (muls $(32, x, y, 32), 32)$
= itimes (nat-to-int $(x, 32)$, nat-to-int $(y, 32)$))

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
$(\text{nat-rangep}(x, n) \land (\text{nat-to-int}(x, n) \neq 0))$
$\rightarrow \quad (\text{nat-to-int}(\text{irem}(n, x, n, y, j), n)$
$\quad\quad = \quad \text{iremainder}(\text{nat-to-int}(y, j), \text{nat-to-int}(x, n)))$

```
; 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
$(\text{int-rangep}(\text{iquotient}(\text{nat-to-int}(y, j), \text{nat-to-int}(x, i)), n) \land (n \leq j))$
$\rightarrow \quad (\text{nat-to-int}(\text{iquot}(n, x, i, y, j), n)$
$\quad\quad = \quad \text{iquotient}(\text{nat-to-int}(y, j), \text{nat-to-int}(x, i)))$

EVENT: Disable head-int-crock.

```
; unsigned interpratation of rem.
```

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

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

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

EVENT: Disable int-rangep.

```
; 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-rangep (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-rangep (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)
$\land$ (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-rangep $(x,\ n)$ $\land$ nat-rangep $(y,\ n)$)
$\rightarrow$ (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-rangep $(x,\ n)$ $\land$ nat-rangep $(y,\ n)$)
$\rightarrow$ (sub $(n,\ x,\ y)$
= **if** $y < x$ **then** $(y +$ exp $(2,\ n)) - x$
else $y - x$ **endif**)

THEOREM: adder-nat-la
$(\text{nat-rangep}\,(x,\ n) \wedge \text{nat-rangep}\,(y,\ n) \wedge \text{bitp}\,(c))$
$\rightarrow \quad (\text{adder}\,(n,\ c,\ x,\ y)$
$= \ \textbf{if}\ (c + x + y) < \exp\,(2,\ n)\ \ \textbf{then}\ c + x + y$
$\textbf{else}\ (c + x + y) - \exp\,(2,\ n)\ \textbf{endif})$

THEOREM: subtracter-nat-la
$(\text{nat-rangep}\,(x,\ n) \wedge \text{nat-rangep}\,(y,\ n) \wedge \text{bitp}\,(c))$
$\rightarrow \quad (\text{subtracter}\,(n,\ c,\ x,\ y)$
$= \ \textbf{if}\ y < (c + x)\ \ \textbf{then}\ (y + \exp\,(2,\ n)) - (c + x)$
$\textbf{else}\ y - (c + x)\ \textbf{endif})$

THEOREM: plus-numberp
$(y \notin \mathbf{N}) \rightarrow ((x + y) = \text{fix}\,(x))$

```
; unsigned interpretation of adder.
```

THEOREM: adder-uint
$(\text{nat-rangep}\,(x,\ n) \wedge \text{nat-rangep}\,(y,\ n) \wedge \text{bitp}\,(c))$
$\rightarrow \quad (\text{nat-to-uint}\,(\text{adder}\,(n,\ c,\ x,\ y))$
$= \ \textbf{if}\ (c + \text{nat-to-uint}\,(x) + \text{nat-to-uint}\,(y)) < \exp\,(2,\ n)$
$\textbf{then}\ c + \text{nat-to-uint}\,(x) + \text{nat-to-uint}\,(y)$
$\textbf{else}\ (c + \text{nat-to-uint}\,(x) + \text{nat-to-uint}\,(y))$
$- \quad \exp\,(2,\ n)\ \textbf{endif})$

```
; unsigned interpretation of subtracter.
```

THEOREM: subtracter-uint
$(\text{nat-rangep}\,(x,\ n) \wedge \text{nat-rangep}\,(y,\ n) \wedge \text{bitp}\,(c))$
$\rightarrow \quad (\text{nat-to-uint}\,(\text{subtracter}\,(n,\ c,\ x,\ y))$
$= \ \textbf{if}\ \text{nat-to-uint}\,(y) < (c + \text{nat-to-uint}\,(x))$
$\textbf{then}\ (\text{nat-to-uint}\,(y) + \exp\,(2,\ n))$
$- \quad (c + \text{nat-to-uint}\,(x))$
$\textbf{else}\ \text{nat-to-uint}\,(y) - (c + \text{nat-to-uint}\,(x))\ \textbf{endif})$

```
; unsigned interpretation of add.
```

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

```
; unsigned interpretation of sub.
```

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

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

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

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

```
; another commutativity theorem about adder.
```

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

```
;                 THEOREMS ABOUT ADD AND SUB
;
; trivial!
```

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

```
; x + 0 = x!
```

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

```
; x - 0 = x!
```

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

```
; x - x = 0!
```

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

```
; 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
$x \not< (x \ \mathbf{mod}\ y)$

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

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

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-rangep
nat-rangep (add $(n,\ x,\ y),\ n)$

THEOREM: sub-nat-rangep
nat-rangep (sub $(n,\ x,\ y),\ n)$

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

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

```
; x + y = x - z <==> y + z = 0.
```

THEOREM: add-sub-cancel
$(\text{nat-rangep}\,(y,\ n) \land \text{nat-rangep}\,(z,\ n))$
$\rightarrow \quad ((\text{add}\,(n,\ x,\ y) = \text{sub}\,(n,\ z,\ x)) = (\text{add}\,(n,\ y,\ z) = \mathtt{0}))$

```
; x - y = 0 <==> x = y.
```

THEOREM: sub-equal-0
$(\text{nat-rangep}\,(x,\ n) \land \text{nat-rangep}\,(y,\ n))$
$\rightarrow \quad ((\text{sub}\,(n,\ x,\ y) = \mathtt{0}) = (\text{fix}\,(x) = \text{fix}\,(y)))$

```
; a bridge from add to adder.
```

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

THEOREM: plus-add1-sub1
$(y \not\simeq \mathtt{0}) \rightarrow ((1 + (x + (y - 1))) = (x + y))$

```
; a bridge from sub to adder.
```

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

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

```
; 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 <==> y = 0.
```

THEOREM: add-cancel0
$(\text{add}(n, x, y) = x) = ((x \in \mathbf{N}) \wedge \text{nat-rangep}(x, n) \wedge (\text{head}(y, n) = \texttt{0}))$

```
; x - y = x <==> y = 0.
```

THEOREM: sub-cancel0
$(\text{nat-rangep}(x, n) \wedge \text{nat-rangep}(y, n))$
$\rightarrow \quad ((\text{sub}(n, y, x) = x) = ((x \in \mathbf{N}) \wedge (y \simeq \texttt{0})))$

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 <==> 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 <==> y = z.
```

THEOREM: sub-cancel
$(\text{nat-rangep}(y, n) \wedge \text{nat-rangep}(z, n))$
$\rightarrow \quad ((\text{sub}(n, y, x) = \text{sub}(n, z, x)) = (\text{fix}(y) = \text{fix}(z)))$

EVENT: Disable add-adder.

EVENT: Disable sub-adder.

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

THEOREM: addy-y
nat-rangep $(x,\ n) \rightarrow (\text{add}\,(n,\ x,\ \text{sub}\,(n,\ y,\ y)) = \text{fix}\,(x))$

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

THEOREM: addy-y1
nat-rangep $(x,\ n) \rightarrow (\text{add}\,(n,\ y,\ \text{sub}\,(n,\ y,\ x)) = \text{fix}\,(x))$

```
;            INTEGER INTERPRETATION OF ADDITION AND SUBTRACTION
```

```
; a bridge for adder-int.
```

THEOREM: adder-int-bridge
(nat-rangep $(x,\ n) \land$ nat-rangep $(y,\ n) \land$ bitp (c))
$\rightarrow$ (nat-to-int (adder $(n,\ c,\ x,\ y),\ n)$
$=$ adder-int-end $(n,\ c,$ nat-to-int $(x,\ n),$ nat-to-int $(y,\ n)))$

EVENT: Disable adder-int-end.

EVENT: Disable adder.

```
; integer interpretation of adder.
```

THEOREM: adder-int
(nat-rangep $(x,\ n) \land$ nat-rangep $(y,\ n) \land$ bitp (c))
$\rightarrow$ (nat-to-int (adder $(n,\ c,\ x,\ y),\ n)$
$=$ **if** int-rangep (iplus (nat-to-int $(x,\ n)$, iplus (nat-to-int $(y,\ n),\ c)$),
n)
then iplus (nat-to-int $(x,\ n)$, iplus (nat-to-int $(y,\ n),\ c)$)
elseif negativep (nat-to-int $(x,\ n)$)
then iplus (nat-to-int $(x,\ n)$,
iplus (nat-to-int $(y,\ n)$, iplus $(c,\ \exp(2,\ n)))$)
else iplus (nat-to-int $(x,\ n)$,
iplus (nat-to-int $(y,\ n)$, iplus $(c,\ -\exp(2,\ n))))$ **endif**)

```
; integer interpretation of subtracter.
```

THEOREM: subtracter-int
$$
\begin{array}{l}
(\text{nat-rangep}\,(x,\ n) \land \text{nat-rangep}\,(y,\ n) \land \text{bitp}\,(c) \land (n \not\simeq \mathtt{0})) \\
\to\ \ (\text{nat-to-int}\,(\text{subtracter}\,(n,\ c,\ x,\ y),\ n) \\
\quad = \ \ \textbf{if}\ \text{int-rangep}\,(\text{idifference}\,(\text{nat-to-int}\,(y,\ n), \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{iplus}\,(\text{nat-to-int}\,(x,\ n),\ c)), \\
\qquad\qquad\qquad\quad n) \\
\qquad\quad \textbf{then}\ \text{idifference}\,(\text{nat-to-int}\,(y,\ n),\ \text{iplus}\,(\text{nat-to-int}\,(x,\ n),\ c)) \\
\qquad\quad \textbf{elseif}\ \text{negativep}\,(\text{nat-to-int}\,(y,\ n)) \\
\qquad\quad \textbf{then}\ \text{idifference}\,(\text{iplus}\,(\text{nat-to-int}\,(y,\ n),\ \exp\,(\mathtt{2},\ n)), \\
\qquad\qquad\qquad\qquad \text{iplus}\,(\text{nat-to-int}\,(x,\ n),\ c)) \\
\qquad\quad \textbf{else}\ \text{idifference}\,(\text{iplus}\,(\text{nat-to-int}\,(y,\ n),\ -\exp\,(\mathtt{2},\ n)), \\
\qquad\qquad\qquad\qquad \text{iplus}\,(\text{nat-to-int}\,(x,\ n),\ c))\ \textbf{endif})
\end{array}
$$

EVENT: Disable subtracter.

```
; integer interpretation of add.
```

THEOREM: add-int
$$
\begin{array}{l}
(\text{nat-rangep}\,(x,\ n) \land \text{nat-rangep}\,(y,\ n)) \\
\to\ \ (\text{nat-to-int}\,(\text{add}\,(n,\ x,\ y),\ n) \\
\quad = \ \ \textbf{if}\ \text{int-rangep}\,(\text{iplus}\,(\text{nat-to-int}\,(x,\ n),\ \text{nat-to-int}\,(y,\ n)),\ n) \\
\qquad\quad \textbf{then}\ \text{iplus}\,(\text{nat-to-int}\,(x,\ n),\ \text{nat-to-int}\,(y,\ n)) \\
\qquad\quad \textbf{elseif}\ \text{negativep}\,(\text{nat-to-int}\,(x,\ n)) \\
\qquad\quad \textbf{then}\ \text{iplus}\,(\text{nat-to-int}\,(x,\ n),\ \text{iplus}\,(\text{nat-to-int}\,(y,\ n),\ \exp\,(\mathtt{2},\ n))) \\
\qquad\quad \textbf{else}\ \text{iplus}\,(\text{nat-to-int}\,(x,\ n), \\
\qquad\qquad\qquad \text{iplus}\,(\text{nat-to-int}\,(y,\ n),\ -\exp\,(\mathtt{2},\ n)))\ \textbf{endif})
\end{array}
$$

```
; integer interpretation of sub.
```

THEOREM: sub-int
$$
\begin{array}{l}
(\text{nat-rangep}\,(x,\ n) \land \text{nat-rangep}\,(y,\ n) \land (n \not\simeq \mathtt{0})) \\
\to\ \ (\text{nat-to-int}\,(\text{add}\,(n,\ x,\ \text{neg}\,(n,\ y)),\ n) \\
\quad = \ \ \textbf{if}\ \text{int-rangep}\,(\text{idifference}\,(\text{nat-to-int}\,(x,\ n),\ \text{nat-to-int}\,(y,\ n)),\ n) \\
\qquad\quad \textbf{then}\ \text{idifference}\,(\text{nat-to-int}\,(x,\ n),\ \text{nat-to-int}\,(y,\ n)) \\
\qquad\quad \textbf{elseif}\ \text{negativep}\,(\text{nat-to-int}\,(x,\ n)) \\
\qquad\quad \textbf{then}\ \text{idifference}\,(\text{iplus}\,(\text{nat-to-int}\,(x,\ n),\ \exp\,(\mathtt{2},\ n)), \\
\qquad\qquad\qquad\qquad \text{nat-to-int}\,(y,\ n)) \\
\qquad\quad \textbf{else}\ \text{idifference}\,(\text{iplus}\,(\text{nat-to-int}\,(x,\ n),\ -\exp\,(\mathtt{2},\ n)), \\
\qquad\qquad\qquad\qquad \text{nat-to-int}\,(y,\ n))\ \textbf{endif})
\end{array}
$$

EVENT: Disable add-neg-adder.

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

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

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
$(j \leq i)$
$\rightarrow \ (((x + (y * \exp(2, i))) \textbf{ mod } \exp(2, j)) = (x \textbf{ mod } \exp(2, j)))$

THEOREM: remainder2-plus-times-exp2
$(i \not\simeq 0) \rightarrow (((x + (y * \exp(2, i))) \textbf{ mod } 2) = (x \textbf{ mod } 2))$

THEOREM: remainder-plus-times-exp2-2
$((x < \exp(2, i)) \land (i \leq j))$
$\rightarrow \ (((x + (y * \exp(2, i))) \textbf{ mod } \exp(2, j))$
$\quad = \ \textbf{if } i < j \textbf{ then } x + ((y \textbf{ mod } \exp(2, j - i)) * \exp(2, i))$
$\quad\quad \textbf{else } \text{fix}(x) \textbf{ endif})$

THEOREM: quotient-plus-times-exp2-1
$(j < i)$
$\rightarrow \ (((x + (y * \exp(2, i))) \div \exp(2, j))$
$\quad = \ ((x \div \exp(2, j)) + (y * \exp(2, i - j))))$

THEOREM: exp2-leq
$(i \leq j) \rightarrow (\exp(2, j) \not< \exp(2, i))$

THEOREM: quotient-plus-times-exp2-2
$((i \leq j) \land (x < \exp(2, i)))$
$\rightarrow \ (((x + (y * \exp(2, i))) \div \exp(2, j)) = (y \div \exp(2, j - i)))$

THEOREM: remainder-quotient-exp2
$((x \textbf{ mod } \exp(2, i)) \div \exp(2, j))$
$= \ \textbf{if } j < i \textbf{ then } (x \div \exp(2, j)) \textbf{ mod } \exp(2, i - j)$
$\quad \textbf{else } 0 \textbf{ endif}$

EVENT: Disable exp2-leq.

THEOREM: head-replace
head (replace (m, x, y), n)
$=$ **if** $m < n$ **then** replace (m, x, head (y, n))
else head (x, n) **endif**

THEOREM: tail-head
tail (head (x, m), n) $=$ head (tail (x, n), $m - n$)

THEOREM: tail-app
tail (app (i, x, y), j)
$=$ **if** $j < i$ **then** app ($i - j$, tail (x, j), y)
else tail (y, $j - i$) **endif**

THEOREM: tail-replace
tail (replace (i, x, y), j)
$=$ **if** $j < i$ **then** replace ($i - j$, tail (x, j), tail (y, j))
else tail (y, j) **endif**

THEOREM: replace-reflex
replace (n, x, x) $=$ fix (x)

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

THEOREM: replace-associativity
replace (n, x, replace (n, y, z)) $=$ replace (n, replace (n, x, y), z)

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

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

THEOREM: head-app
$(i \leq j) \rightarrow$ (head (app (j, x, y), i) $=$ 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
$(\mathrm{bcar}\,(2 * x) = 0) \land (\mathrm{bcar}\,(1 + (2 * x)) = 1)$

THEOREM: bcdr-1
$(\mathrm{bcdr}\,(2 * x) = \mathrm{fix}\,(x)) \land (\mathrm{bcdr}\,(1 + (2 * x)) = \mathrm{fix}\,(x))$

THEOREM: bcar-2
$\mathrm{bcar}\,(x + (2 * y)) = \mathrm{bcar}\,(x)$

THEOREM: bcdr-2
$\mathrm{bcdr}\,(x + (2 * y)) = (\mathrm{bcdr}\,(x) + y)$

```
; the first "bit" of head.
```

THEOREM: bcar-head
$\mathrm{bcar}\,(\mathrm{head}\,(x,\ n))$
$=$ **if** $n \simeq 0$ **then** 0
else $\mathrm{bcar}\,(x)$ **endif**

```
; the first "bit" of app.
```

THEOREM: bcar-app
$\mathrm{bcar}\,(\mathrm{app}\,(n,\ x,\ y))$
$=$ **if** $n \simeq 0$ **then** $\mathrm{bcar}\,(y)$
else $\mathrm{bcar}\,(x)$ **endif**

```
; the first "bit" of replace.
```

THEOREM: bcar-replace
$\mathrm{bcar}\,(\mathrm{replace}\,(n,\ x,\ y))$
$=$ **if** $n \simeq 0$ **then** $\mathrm{bcar}\,(y)$
else $\mathrm{bcar}\,(x)$ **endif**

THEOREM: bcar-lessp
$\mathrm{bcar}\,(x) < 2$

THEOREM: bcdr-lessp
$(\mathrm{bcdr}\,(x) < y) = (x < (2 * y))$

THEOREM: app-associativity
$\mathrm{app}\,(n1,\ x,\ \mathrm{app}\,(n2,\ y,\ z)) = \mathrm{app}\,(n1 + n2,\ \mathrm{app}\,(n1,\ x,\ y),\ z)$

THEOREM: app-head-tail
$\mathrm{app}\,(n,\ \mathrm{head}\,(x,\ n),\ \mathrm{tail}\,(x,\ n)) = \mathrm{fix}\,(x)$

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

EVENT: Disable head.

EVENT: Disable tail.

EVENT: Disable bcar.

EVENT: Disable bcdr.

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

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

EVENT: Disable app.

EVENT: Disable replace.

```
;                    BITN
```

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

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

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

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

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

THEOREM: bitn-replace
$\text{bitn}\,(\text{replace}\,(n,\ x,\ y),\ k)$
$=$ **if** $k < n$ **then** $\text{bitn}\,(x,\ k)$
else $\text{bitn}\,(y,\ k)$ **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 $(x, y) = 0$) $=$ (fix (x) $=$ fix (y))

THEOREM: logand-commutativity
logand (x, y) $=$ logand (y, x)

THEOREM: logor-commutativity
logor (x, y) $=$ logor (y, x)

THEOREM: logeor-commutativity
logeor (x, y) $=$ logeor (y, x)

THEOREM: logand-logor
logand $(x$, logor $(y, z))$ $=$ logor (logand (x, y), logand (x, z))

THEOREM: logand-logeor
logand $(x$, logeor $(y, z))$ $=$ logeor (logand (x, y), logand (x, z))

THEOREM: logand-uint-la
nat-to-uint (logand (exp $(2, i) - 1, y$)) $=$ (nat-to-uint (y) **mod** exp $(2, i)$)

THEOREM: logand-uint
(exp (2, log $(2, 1 + x)$) $= (1 + x)$)
$\rightarrow$ (nat-to-uint (logand (x, y)) $=$ (nat-to-uint (y) **mod** $(1 + x)$))

```
;             HEAD WITH READ/WRITE AND READM/WRITEM
```

THEOREM: head-readp
$(n \leq k) \rightarrow$ (readp (head (x, k), n, *map*) $=$ readp $(x, n,$ *map*$)$)

THEOREM: head-pc-readp
$(n \leq k) \rightarrow$ (pc-readp (head (x, k), n, *map*) $=$ pc-readp $(x, n,$ *map*$)$)

THEOREM: head-writep
$(n \le k) \rightarrow (\text{writep}\,(\text{head}\,(x,\ k),\ n,\ \mathit{map}) = \text{writep}\,(x,\ n,\ \mathit{map}))$

THEOREM: head-read
$(n \le k) \rightarrow (\text{read}\,(\text{head}\,(x,\ k),\ n,\ \mathit{bt}) = \text{read}\,(x,\ n,\ \mathit{bt}))$

THEOREM: head-pc-read
$(n \le k) \rightarrow (\text{pc-read}\,(\text{head}\,(x,\ k),\ n,\ \mathit{bt}) = \text{pc-read}\,(x,\ n,\ \mathit{bt}))$

THEOREM: head-write
$(n \le k) \rightarrow (\text{write}\,(\mathit{value},\ \text{head}\,(x,\ k),\ n,\ \mathit{bt}) = \text{write}\,(\mathit{value},\ x,\ n,\ \mathit{bt}))$

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

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

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

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

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

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

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

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

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

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

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

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

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

; NAT-RANGEP

THEOREM: nat-rangep-of-0
nat-rangep $(\texttt{0},\ n)$

THEOREM: nat-rangep-ub
nat-rangep $(\exp(\texttt{2},\ k) - 1,\ n) = (n \not< k)$

THEOREM: nat-rangep-0
nat-rangep $(x,\ \texttt{0}) = (x \simeq \texttt{0})$

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

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

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

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

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

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

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

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

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

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

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

THEOREM: read-rn-nat-rangep
nat-rangep (read-rn (n, rn, $rfile$), n)

THEOREM: byte-read-nat-rangep
$(8 \le n) \rightarrow$ nat-rangep (byte-read (x, mem), n)

THEOREM: pc-byte-read-nat-rangep
nat-rangep (pc-byte-read (x, mem), 8)

EVENT: Disable nat-rangep.

THEOREM: mulu-nat-rangep
nat-rangep (mulu (n, x, y, i), n)

THEOREM: muls-nat-rangep
nat-rangep (muls (n, x, y, i), n)

THEOREM: quot-nat-rangep
nat-rangep (quot (n, x, y), n)

THEOREM: rem-nat-rangep
nat-rangep (rem (n, x, y), n)

THEOREM: iquot-nat-rangep
nat-rangep (iquot (n, x, i, y, j), n)

THEOREM: irem-nat-rangep
nat-rangep (irem (n, x, i, y, j), n)

THEOREM: neg-nat-rangep
nat-rangep (neg (n, x), n)

; EXT

THEOREM: ext-0
ext (n, 0, $size$) $=$ 0

THEOREM: ext-lemma
$(size \le n) \rightarrow$ (ext (n, x, $size$) $=$ head (x, $size$))

THEOREM: head-ext
$((i \le j) \land (j \le k)) \rightarrow$ (head (ext (j, x, k), i) $=$ head (x, i))

THEOREM: ext-nat-rangep
nat-rangep (ext (n, x, $size$), $size$)

EVENT: Disable ext.

`;                                    CVZNX-FLAGS`

THEOREM: set-cvznx-c
ccr-c (set-cvznx (cvznx (c, v, z, n, x), ccr)) = fix-bit (c)

THEOREM: set-cvznx-v
ccr-v (set-cvznx (cvznx (c, v, z, n, x), ccr)) = fix-bit (v)

THEOREM: set-cvznx-z
ccr-z (set-cvznx (cvznx (c, v, z, n, x), ccr)) = fix-bit (z)

THEOREM: set-cvznx-n
ccr-n (set-cvznx (cvznx (c, v, z, n, x), ccr)) = fix-bit (n)

THEOREM: set-cvznx-x
ccr-x (set-cvznx (cvznx (c, v, z, n, x), ccr)) = fix-bit (x)

`; the new cvznx-flags replaces the old ones.`

THEOREM: set-set-cvznx1
set-cvznx (x, set-cvznx (y, ccr)) = set-cvznx (x, ccr)

THEOREM: set-cvznx-ccr
set-cvznx (ccr, ccr) = fix (ccr)

THEOREM: set-set-cvznx2
set-cvznx (set-cvznx (x, ccr), ccr) = set-cvznx (x, ccr)

THEOREM: set-cvznx-nat-rangep
nat-rangep (set-cvznx (x, ccr), 8) = nat-rangep (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-rangep (*pc*, 32)
$\rightarrow$ (mc-pc (mc-state (*status*, *rfile*, *pc*, *ccr*, *mem*)) = fix (*pc*))

THEOREM: mc-ccr-rewrite
nat-rangep (*ccr*, 8)
$\rightarrow$ (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-rangep
nat-rangep (mc-pc (*s*), 32)

THEOREM: mc-ccr-rangep
nat-rangep (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
$(i \notin \mathbf{N}) \rightarrow$ (get-nth (i, *lst*) = get-nth (0, *lst*))

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

EVENT: Disable get-nth-0.

EVENT: Disable put-nth-0.

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

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

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

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

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

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

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

```
; stop proving.
```

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

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

THEOREM: write-write-rn
$(n1 \le n2)$
$\rightarrow$ (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* < 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* $\simeq$ **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 $(n1 \le n2) \wedge (n2 \le \texttt{32})$
then get-vlst $(n1, \texttt{0}, rn, rnlst, vlst)$
else read-writem-rn-end $(n1, rn, n2, vlst, rnlst, rfile)$ **endif**
else read-rn $(n1, rn, rfile)$ **endif**

EVENT: Disable read-writem-rn-end.

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

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

THEOREM: get-vlst-readm-rn
$(n1 \le n2)$
$\rightarrow$ (get-vlst $(n1, \texttt{0}, rn, rnlst,$ readm-rn $(n2, rnlst, rfile))$
$=$ **if** n-member $(rn, rnlst)$ **then** read-rn $(n1, rn, rfile)$
else 0 **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
pc-readp $(x, n, map) \rightarrow$ readp (x, n, map)

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

```
; a read right after a write at the same location returns the new value
; just writen.  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:
modn-eq (n, x, y)
$=$ **if** $n \simeq \texttt{0}$ **then t**
else (bitn $(x, n-1) =$ bitn $(y, n-1)) \wedge$ modn-eq $(n-1, x, y)$ **endif**

THEOREM: read-write
$\text{read}(y, n, \text{write}(value, x, n, bt))$
$= \ \textbf{if} \ \text{modn-eq}(n, x, y) \ \textbf{then} \ value$
$\quad \textbf{else} \ \text{read}(y, n, bt) \ \textbf{endif}$

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

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

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

THEOREM: write-write-la
$\text{write}(v2, y, n, \text{write}(v1, x, n, bt))$
$= \ \textbf{if} \ \neg \ \text{modn-eq}(n, x, y) \ \textbf{then} \ \text{write}(v1, x, n, \text{write}(v2, y, n, bt))$
$\quad \textbf{else} \ \text{write}(v2, y, n, bt) \ \textbf{endif}$

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

EVENT: Disable pc-read.

THEOREM: plus-times-equal
$((a < k) \wedge (b < k))$
$\rightarrow \ (((a + (i * k)) = (b + (j * k)))$
$\quad = \ ((\text{fix}(a) = \text{fix}(b)) \wedge (\text{fix}(i) = \text{fix}(j))))$

THEOREM: app-cancel
$(\text{app}(n, x, y) = \text{app}(n, x1, y1))$
$= \ ((\text{head}(x, n) = \text{head}(x1, n)) \wedge (\text{fix}(y) = \text{fix}(y1)))$

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
$((n \leq \mathit{size}) \land (n \not\simeq 0))$
$\rightarrow \quad ((\mathrm{ext}(n,\ x,\ \mathit{size}) = \mathrm{ext}(n,\ y,\ \mathit{size})) = (\mathrm{head}(x,\ n) = \mathrm{head}(y,\ n)))$

THEOREM: ext-equal-0
$((n \leq \mathit{size}) \land (n \not\simeq 0))$
$\rightarrow \quad ((\mathrm{ext}(n,\ x,\ \mathit{size}) = 0) = (\mathrm{head}(x,\ n) = 0))$

EVENT: Disable head-recursion.

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

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

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

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

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

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

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

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

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

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

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

THEOREM: byte-write-maintain-byte-writep
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:
mem-induct0 $(k,\ n)$
$=$ **if** $k \simeq 0$ **then** 0
else mem-induct0 $(k - 1,\ n - 8)$ **endif**

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

THEOREM: pc-read-mem-write-mem
(write-memp (x, mem, m) $\land$ pc-read-memp (y, mem, n))
$\rightarrow$ (pc-read-mem (y, write-mem ($value$, x, mem, m), n) = pc-read-mem (y, 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
$((\text{head}(x, n) - 1) < x) = (x \not\simeq 0)$

THEOREM: byte-read-write-mem-lemma
$(\text{nat-rangep}(y, 32) \land (n \leq y))$
$\rightarrow$ (byte-read (add (32, x, y), write-mem ($value$, x, mem, n))
$=$ byte-read (add (32, x, y), mem))

THEOREM: read-write-mem2
uint-rangep (n, 32)
$\rightarrow$ (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, byte-write ($v1$, x, mem), n)
$=$ **if** disjoint0 (y, n, x) **then** byte-write ($v1$, x, write-mem ($v2$, y, mem, n))
else write-mem ($v2$, y, mem, n) **endif**

DEFINITION:
write-write-induct ($v1$, $v2$, n)
$=$ **if** $n \simeq 0$ **then t**
else write-write-induct (tail ($v1$, B), tail ($v2$, B), $n - 1$) **endif**

THEOREM: write-write-mem
write-mem ($v2$, x, write-mem ($v1$, x, mem, n), n) $=$ write-mem ($v2$, x, 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
bitp (lsl-c (len, x, cnt))

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

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

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

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

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

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

THEOREM: asl-v-bitp
bitp (asl-v (len, x, cnt))

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

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

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

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

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

THEOREM: roxl-c-bitp
bitp (roxl-c (len, opd, cnt, x))

THEOREM: roxl-z-bitp
bitp (roxl-z (len, opd, cnt, x))

THEOREM: roxl-n-bitp
bitp (roxl-n (len, opd, cnt, x))

THEOREM: roxr-c-bitp
bitp (roxr-c (len, opd, cnt, x))

THEOREM: roxr-z-bitp
bitp (roxr-z (len, opd, cnt, x))

THEOREM: roxr-n-bitp
bitp (roxr-n (len, opd, cnt, x))

THEOREM: move-z-bitp
bitp (move-z ($oplen$, $sopd$))

THEOREM: move-n-bitp
bitp (move-n ($oplen$, $sopd$))

THEOREM: ext-z-bitp
bitp (ext-z ($oplen$, opd, $size$))

THEOREM: ext-n-bitp
bitp (ext-n ($oplen$, opd, $size$))

THEOREM: swap-z-bitp
bitp (swap-z (opd))

THEOREM: swap-n-bitp
bitp (swap-n (opd))

THEOREM: not-z-bitp
bitp (not-z ($oplen$, opd))

THEOREM: not-n-bitp
bitp (not-n ($oplen$, opd))

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

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

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

THEOREM: neg-head
neg (n, head (x, n)) = neg (n, x)

THEOREM: neg-neg
neg (n, neg (n, x)) = head (x, n)

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

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 <==> x = y.
```

THEOREM: add-neg-0
$(\text{nat-rangep}\,(x,\ n) \land \text{nat-rangep}\,(y,\ n))$
$\rightarrow \quad ((\text{add}\,(n,\ y,\ \text{neg}\,(n,\ x)) = \mathtt{0}) = (\text{fix}\,(x) = \text{fix}\,(y)))$

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

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

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

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

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

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

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

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
$$
\begin{array}{l}
(e \in x) \\
\rightarrow \quad (\text{eval\$}(\mathbf{t},\ \text{add-tree}(n,\ x),\ a) \\
\qquad = \quad \text{add}(\text{eval\$}(\mathbf{t},\ n,\ a), \\
\qquad\qquad\qquad \text{eval\$}(\mathbf{t},\ e,\ a), \\
\qquad\qquad\qquad \text{eval\$}(\mathbf{t},\ \text{add-tree}(n,\ \text{delete}(e,\ x)),\ a)))
\end{array}
$$

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

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

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:
$$
\begin{array}{l}
\text{add-tree-delete-cond}(n,\ x,\ y) \\
= \quad \text{list}(\texttt{'and}, \\
\qquad\qquad \text{list}(\texttt{'numberp},\ x), \\
\qquad\qquad \text{list}(\texttt{'and}, \\
\qquad\qquad\qquad \text{list}(\texttt{'nat-rangep},\ x,\ n), \\
\qquad\qquad\qquad \text{list}(\texttt{'equal},\ \text{add-tree}(n,\ \text{delete}(x,\ y)),\ \texttt{''0})))
\end{array}
$$

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

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

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

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

DEFINITION:
cancel-add-neg (x)
$=$ **if** listp (x)
$\land$ (car (x) = `'add`)
$\land$ listp (cdr (x))
$\land$ listp (cddr (x))
then if listp (caddr (x))
$\land$ listp (cdaddr (x))
$\land$ listp (cddaddr (x))
$\land$ (caaddr (x) = `'neg`)
$\land$ (cadaddr (x) = cadr (x))
$\land$ (caddaddr (x) $\in$ add-fringe (cadr (x), cadddr (x)))
then add-tree (cadr (x),
delete (caddaddr (x), add-fringe (cadr (x), cadddr (x))))
elseif list (`'neg`, cadr (x), caddr (x))
$\in$ add-fringe (cadr (x), cadddr (x))
then add-tree (cadr (x),
delete (list (`'neg`, cadr (x), caddr (x)),
add-fringe (cadr (x), cadddr (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
sub-c $(n,\ x,\ y) =$ subx-c $(n,\ \mathtt{0},\ x,\ y)$

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

THEOREM: addx-c-la
(nat-rangep $(x,\ n) \land$ nat-rangep $(y,\ n) \land$ bitp $(c) \land (n \not\simeq \mathtt{0}))$
$\rightarrow$ (addx-c $(n,\ c,\ x,\ y)$
$=$ **if** $(c + x + y) <$ exp $(\mathtt{2},\ n)$ **then** $\mathtt{0}$
else $\mathtt{1}$ **endif**)

THEOREM: subx-c-la
(nat-rangep $(x,\ n) \land$ nat-rangep $(y,\ n) \land$ bitp $(c) \land (n \not\simeq \mathtt{0}))$
$\rightarrow$ (subx-c $(n,\ c,\ x,\ y)$
$=$ **if** $y < (c + x)$ **then** $\mathtt{1}$
else $\mathtt{0}$ **endif**)

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
(nat-rangep $(x,\ n) \land$ nat-rangep $(y,\ n))$
$\rightarrow$ (sub-z $(n,\ x,\ y)$
$=$ **if** fix $(x) =$ fix (y) **then** $\mathtt{1}$
else $\mathtt{0}$ **endif**)

DEFINITION: between-ileq $(x,\ y,\ z) =$ (ileq $(x,\ y) \land$ ileq $(y,\ z))$

THEOREM: sub-bhi-int
(nat-rangep $(x,\ n)$
$\land$ nat-rangep $(y,\ n)$
$\land$ $(n \not\simeq \mathtt{0})$
$\land$ $(\neg$ negativep (nat-to-int $(x,\ n))))$
$\rightarrow$ (bhi (sub-c $(n,\ x,\ y)$, sub-z $(n,\ x,\ y))$
$=$ **if** between-ileq ($\mathtt{0}$, nat-to-int $(y,\ n)$, nat-to-int $(x,\ n))$ **then** $\mathtt{0}$
else $\mathtt{1}$ **endif**)

THEOREM: sub-bhi-0
$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{nat-rangep}\,(x,\ n)} \\
\ \wedge & \text{nat-rangep}\,(y,\ n) \\
\ \wedge & (\text{nat-to-uint}\,(x) \not< \text{nat-to-uint}\,(y)) \\
\ \wedge & (n \not\simeq \mathtt{0})) \\
\rightarrow & (\text{bhi}\,(\text{sub-c}\,(n,\ x,\ y),\ \text{sub-z}\,(n,\ x,\ y)) = \mathtt{0})
\end{array}
$$

THEOREM: sub-bhi-1
$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{nat-rangep}\,(x,\ n)} \\
\ \wedge & \text{nat-rangep}\,(y,\ n) \\
\ \wedge & (\text{nat-to-uint}\,(x) < \text{nat-to-uint}\,(y)) \\
\ \wedge & (n \not\simeq \mathtt{0})) \\
\rightarrow & (\text{bhi}\,(\text{sub-c}\,(n,\ x,\ y),\ \text{sub-z}\,(n,\ x,\ y)) = \mathtt{1})
\end{array}
$$

THEOREM: sub-bls
$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{nat-rangep}\,(x,\ n) \wedge \text{nat-rangep}\,(y,\ n) \wedge (n \not\simeq \mathtt{0}))} \\
\rightarrow & (\text{bls}\,(\text{sub-c}\,(n,\ x,\ y),\ \text{sub-z}\,(n,\ x,\ y)) \\
& \quad = \ \textbf{if}\ \text{nat-to-uint}\,(x) < \text{nat-to-uint}\,(y)\ \ \textbf{then}\ \mathtt{0} \\
& \qquad\ \ \textbf{else}\ \mathtt{1}\ \textbf{endif})
\end{array}
$$

```
; 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
$$
\begin{array}{ll}
\multicolumn{2}{l}{\text{nat-rangep}\,(x,\ n)} \\
\rightarrow & (\text{beq}\,(\text{move-z}\,(n,\ x)) \\
& \quad = \ \textbf{if}\ \text{nat-to-uint}\,(x) = \mathtt{0}\ \ \textbf{then}\ \mathtt{1} \\
& \qquad\ \ \textbf{else}\ \mathtt{0}\ \textbf{endif})
\end{array}
$$

```
; the signed integer interpretation.
```

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

THEOREM: move-beq-int-1
$(\text{nat-rangep}\,(x,\ n) \land (\text{nat-to-int}\,(x,\ n) = 0)) \rightarrow (\text{beq}\,(\text{move-z}\,(n,\ x)) = 1)$

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

THEOREM: move-beq-ext
$(\text{nat-rangep}\,(x,\ n) \land (n \leq \mathit{size}) \land (n \not\simeq 0))$
$\rightarrow \quad (\text{move-z}\,(\mathit{size},\ \text{ext}\,(n,\ x,\ \mathit{size})) = \text{move-z}\,(n,\ x))$

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

THEOREM: sub-beq-uint
$(\text{nat-rangep}\,(x,\ n) \land \text{nat-rangep}\,(y,\ n))$
$\rightarrow \quad (\text{beq}\,(\text{sub-z}\,(n,\ x,\ y))$
$\quad\quad = \quad \textbf{if}\ \text{nat-to-uint}\,(x) = \text{nat-to-uint}\,(y)\ \ \textbf{then}\ 1$
$\quad\quad\quad\quad \textbf{else}\ 0\ \textbf{endif})$

```
; the signed integer interpretation.
```

THEOREM: sub-beq-int-0
$(\text{nat-rangep}\,(x,\ n)$
$\land \quad \text{nat-rangep}\,(y,\ n)$
$\land \quad (\text{nat-to-int}\,(x,\ n) \neq \text{nat-to-int}\,(y,\ n)))$
$\rightarrow \quad (\text{beq}\,(\text{sub-z}\,(n,\ x,\ y)) = 0)$

THEOREM: sub-beq-int-1
$(\text{nat-rangep}\,(x,\ n)$
$\land \quad \text{nat-rangep}\,(y,\ n)$
$\land \quad (\text{nat-to-int}\,(x,\ n) = \text{nat-to-int}\,(y,\ n)))$
$\rightarrow \quad (\text{beq}\,(\text{sub-z}\,(n,\ x,\ y)) = 1)$

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

THEOREM: sub-beq-ext
$(\text{nat-rangep}\,(x,\ n) \land \text{nat-rangep}\,(y,\ n) \land (n \leq \mathit{size}) \land (n \not\simeq 0))$
$\rightarrow \quad (\text{sub-z}\,(\mathit{size},\ \text{ext}\,(n,\ x,\ \mathit{size}),\ \text{ext}\,(n,\ y,\ \mathit{size})) = \text{sub-z}\,(n,\ x,\ y))$

```
; the z flag of logor.
```

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

THEOREM: logor-beq-int-0
$$(\text{nat-rangep}(x,\ n) \land \text{nat-rangep}(y,\ n) \land ((\text{nat-to-int}(x,\ n) \neq 0) \lor (\text{nat-to-int}(y,\ n) \neq 0))) \rightarrow (\text{beq}(\text{or-z}(n,\ x,\ y)) = 0)$$

THEOREM: logor-beq-int-1
$$(\text{nat-rangep}(x,\ n) \land \text{nat-rangep}(y,\ n) \land (\text{nat-to-int}(x,\ n) = 0) \land (\text{nat-to-int}(y,\ n) = 0)) \rightarrow (\text{beq}(\text{or-z}(n,\ x,\ y)) = 1)$$

```
; the z flag of logeor.
```

THEOREM: logeor-beq-uint
$$\text{beq}(\text{eor-z}(n,\ x,\ y)) = \textbf{if}\ \text{nat-to-uint}(x) = \text{nat-to-uint}(y)\ \textbf{then}\ 1\ \textbf{else}\ 0\ \textbf{endif}$$

THEOREM: logeor-beq-int-0
$$(\text{nat-rangep}(x,\ n) \land \text{nat-rangep}(y,\ n) \land (\text{nat-to-int}(x,\ n) \neq \text{nat-to-int}(y,\ n))) \rightarrow (\text{beq}(\text{eor-z}(n,\ x,\ y)) = 0)$$

THEOREM: logeor-beq-int-1
$$(\text{nat-rangep}(x,\ n) \land \text{nat-rangep}(y,\ n) \land (\text{nat-to-int}(x,\ n) = \text{nat-to-int}(y,\ n))) \rightarrow (\text{beq}(\text{eor-z}(n,\ x,\ y)) = 1)$$

```
; the z flag of logand.
```

THEOREM: logand-beq-uint
$$(\exp(2,\ \log(2,\ 1+x)) = (1+x)) \rightarrow (\text{beq}(\text{and-z}(n,\ x,\ y)) = \textbf{if}\ (\text{nat-to-uint}(y)\ \textbf{mod}\ (1+x)) = 0\ \textbf{then}\ 1\ \textbf{else}\ 0\ \textbf{endif})$$

THEOREM: logand-or-beq-uint
$$(\exp(2,\ \log(2,\ 1+x)) = (1+x)) \rightarrow (\text{beq}(\text{and-z}(n,\ x,\ \text{logor}(y,\ z))) = \textbf{if}\ ((\text{nat-to-uint}(y)\ \textbf{mod}\ (1+x)) = 0) \land ((\text{nat-to-uint}(z)\ \textbf{mod}\ (1+x)) = 0)\ \textbf{then}\ 1\ \textbf{else}\ 0\ \textbf{endif})$$

THEOREM: logand-eor-beq-uint
$(\exp(2, \log(2, 1 + x)) = (1 + x))$
$\rightarrow$ (beq (and-z $(n, x, \text{logeor}(y, z))$)
= **if** (nat-to-uint (y) **mod** $(1 + x)$)
= (nat-to-uint (z) **mod** $(1 + x)$) **then** 1
else 0 **endif**)

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

THEOREM: mulu_1632-beq
(nat-rangep $(x, 16)$ $\wedge$ nat-rangep $(y, 16)$)
$\rightarrow$ (beq (mulu-z $(32, x, y, 16)$)
= **if** (nat-to-uint $(x) = 0$) $\vee$ (nat-to-uint $(y) = 0$) **then** 1
else 0 **endif**)

THEOREM: mulu_3264-beq
(nat-rangep $(x, 32)$ $\wedge$ nat-rangep $(y, 32)$)
$\rightarrow$ (beq (mulu-z $(64, x, y, 32)$)
= **if** (nat-to-uint $(x) = 0$) $\vee$ (nat-to-uint $(y) = 0$) **then** 1
else 0 **endif**)

THEOREM: mulu_3232-beq
uint-rangep (nat-to-uint (x) $*$ nat-to-uint (y), 32)
$\rightarrow$ (beq (mulu-z $(32, x, y, 32)$)
= **if** (nat-to-uint $(x) = 0$) $\vee$ (nat-to-uint $(y) = 0$) **then** 1
else 0 **endif**)

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

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

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

THEOREM: muls_3232-beq
(int-rangep (itimes (nat-to-int $(x, 32)$, nat-to-int $(y, 32)$), 32)

$\wedge$ nat-rangep $(x,\ 32)$
$\wedge$ nat-rangep $(y,\ 32))$
$\rightarrow$ (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-rangep $(y,\ n) \wedge$ (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-rangep (iquotient (nat-to-int $(y,\ j)$, nat-to-int $(x,\ i)),\ n)$
$\wedge$ nat-rangep $(x,\ i)$
$\wedge$ (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-rangep (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-rangep $(x,\ n)$ $\land$ int-rangep (nat-to-int $(x,\ n)$, $n\ -\ cnt$))
$\rightarrow$ (beq (asl-z $(n,\ x,\ cnt)$)
= **if** nat-to-int $(x,\ n) = \texttt{0}$ **then** `1`
else `0` **endif**)

```
; the z-flag of asr.
```

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

```
; the z-flag of ext.
```

THEOREM: ext-beq-uint
(nat-rangep $(x,\ n)$ $\land$ $(n \leq size)$)
$\rightarrow$ (beq (ext-z $(n,\ x,\ size)$)
= **if** nat-to-uint $(x) = \texttt{0}$ **then** `1`
else `0` **endif**)

THEOREM: ext-beq-int-0
(nat-rangep $(x,\ n)$ $\land$ $(n < size)$ $\land$ (nat-to-int $(x,\ n) \neq \texttt{0}$))
$\rightarrow$ (beq (ext-z $(n,\ x,\ size)$) = `0`)

THEOREM: ext-beq-int-1
(nat-rangep $(x,\ n)$ $\land$ $(n < size)$ $\land$ (nat-to-int $(x,\ n) = \texttt{0}$))
$\rightarrow$ (beq (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
(nat-rangep $(x,\ n)$ $\wedge$ nat-rangep $(y,\ n)$ $\wedge$ $(n \not\simeq \mathtt{0})$)
$\rightarrow$ (bcs (sub-c $(n,\ x,\ y)$)
 = **if** nat-to-uint (y) < nat-to-uint (x) **then** 1
 else 0 **endif**)

```
; the c-flag of add.
```

THEOREM: add-bcs&cc
(nat-rangep $(x,\ n)$ $\wedge$ nat-rangep $(y,\ n)$ $\wedge$ $(n \not\simeq \mathtt{0})$)
$\rightarrow$ (bcs (add-c $(n,\ x,\ y)$)
 = **if** (nat-to-uint (x) + nat-to-uint (y)) < exp $(2,\ n)$ **then** 0
 else 1 **endif**)

THEOREM: tail-mbit
$(x < \exp(2,\ n))$
$\rightarrow$ (tail $(x,\ n - 1)$
 = **if** $x < \exp(2,\ n - 1)$ **then** 0
 else 1 **endif**)

```
; 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
lsl-c $(n,\ x,\ \mathtt{0}) = \mathtt{0}$

THEOREM: lsl-1-bcs&cc
(nat-rangep $(x,\ n)$ $\wedge$ $(\mathtt{1} < n)$)
$\rightarrow$ (bcs (lsl-c $(n,\ x,\ \mathtt{1})$)
 = **if** uint-rangep (nat-to-uint (x), $n - 1$) **then** 0
 else 1 **endif**)

```
; 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
lsr-c $(n,\ x,\ \mathtt{0}) = \mathtt{0}$

THEOREM: lsr-1-bcs&cc
$(\mathtt{1} < n)$ $\rightarrow$ (bcs (lsr-c $(n,\ x,\ \mathtt{1})$) = (nat-to-uint (x) **mod** 2))

EVENT: Disable tail-mbit.

EVENT: Disable bcs.

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

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

THEOREM: subx-addx-v
(nat-rangep $(x,\ n) \land (n \not\simeq \mathtt{0}))$
$\rightarrow$ (subx-v $(n,\ z,\ x,\ y) =$ addx-v $(n,$ b-not $(z),\ y,$ lognot $(n,\ x)))$

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

```
; lemmas for addx-v and subx-v.
```

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

THEOREM: addx-v-la
(nat-rangep $(x,\ n) \land$ nat-rangep $(y,\ n) \land$ bitp $(c) \land (n \not\simeq \mathtt{0}))$
$\rightarrow$ (addx-v $(n,\ c,\ x,\ y)$
$=$ **if** int-rangep (iplus (nat-to-int $(x,\ n)$, iplus (nat-to-int $(y,\ n),\ c)$),
$n)$ **then** $\mathtt{0}$
else $\mathtt{1}$ **endif**)

THEOREM: subx-v-la
(nat-rangep $(x,\ n) \land$ nat-rangep $(y,\ n) \land$ bitp $(c) \land (n \not\simeq \mathtt{0}))$
$\rightarrow$ (subx-v $(n,\ c,\ x,\ y)$
$=$ **if** int-rangep (idifference (nat-to-int $(y,\ n)$,
iplus (nat-to-int $(x,\ n),\ c)$),
$n)$ **then** $\mathtt{0}$
else $\mathtt{1}$ **endif**)

EVENT: Disable addx-v-crock1.

EVENT: Disable addx-v-crock2.

```
; the v-flag of sub.
```

THEOREM: sub-bvs&vc
(nat-rangep (x, n) $\wedge$ nat-rangep (y, n) $\wedge$ $(n \not\simeq 0)$)
$\rightarrow$ (bvs (sub-v (n, x, y))
= **if** int-rangep (idifference (nat-to-int (y, n), nat-to-int (x, n)), n)
then 0
else 1 **endif**)

```
; the v-flag of add.
```

THEOREM: add-bvs&vc
(nat-rangep (x, n) $\wedge$ nat-rangep (y, n) $\wedge$ $(n \not\simeq 0)$)
$\rightarrow$ (bvs (add-v (n, x, y))
= **if** int-rangep (iplus (nat-to-int (x, n), nat-to-int (y, n)), n)
then 0
else 1 **endif**)

```
; three cases for the v-flag of mulu.
```

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

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

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

```
; three cases for the v-flag of muls.
```

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

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

THEOREM: muls_3232-bvs
bvs (muls-v $(32, x, y, 32)$)
= **if** int-rangep (itimes (nat-to-int $(x, 32)$, nat-to-int $(y, 32)$), 32) **then** 0
else 1 **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) ∧ ($n \not\simeq 0$))
→ (bmi (sub-n (n, x, y))
= **if** int-rangep (idifference (nat-to-int (y, n), nat-to-int (x, n)), n)
then if ilessp (nat-to-int (y, n), nat-to-int (x, n)) **then** 1
else 0 **endif**
elseif ilessp (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) ∧ ($n \not\simeq 0$))
→ (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

$$
\begin{array}{l}
\qquad\qquad\quad \textbf{else } \mathtt{0} \textbf{ endif} \\
\qquad\quad \textbf{elseif } \text{negativep}\,(\text{nat-to-int}\,(x,\ n)) \ \textbf{ then } \mathtt{0} \\
\qquad\quad \textbf{else } \mathtt{1} \textbf{ endif})
\end{array}
$$

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

THEOREM: muls_1632-bmi

$$
\begin{array}{lll}
\multicolumn{3}{l}{(\text{nat-rangep}\,(x,\ \mathtt{16}) \land \text{nat-rangep}\,(y,\ \mathtt{16}))} \\
\rightarrow & \multicolumn{2}{l}{(\text{bmi}\,(\text{muls-n}\,(\mathtt{32},\ x,\ y,\ \mathtt{16}))} \\
 & = & \textbf{if } (\text{nat-to-int}\,(x,\ \mathtt{16}) = \mathtt{0}) \lor (\text{nat-to-int}\,(y,\ \mathtt{16}) = \mathtt{0}) \\
 & & \textbf{then } \mathtt{0} \\
 & & \textbf{elseif } \text{negativep}\,(\text{nat-to-int}\,(x,\ \mathtt{16})) \\
 & & \textbf{then if } \text{negativep}\,(\text{nat-to-int}\,(y,\ \mathtt{16})) \ \textbf{ then } \mathtt{0} \\
 & & \qquad \textbf{else } \mathtt{1} \textbf{ endif} \\
 & & \textbf{elseif } \text{negativep}\,(\text{nat-to-int}\,(y,\ \mathtt{16})) \ \textbf{ then } \mathtt{1} \\
 & & \textbf{else } \mathtt{0} \textbf{ endif})
\end{array}
$$

THEOREM: muls_3264-bmi

$$
\begin{array}{lll}
\multicolumn{3}{l}{(\text{nat-rangep}\,(x,\ \mathtt{32}) \land \text{nat-rangep}\,(y,\ \mathtt{32}))} \\
\rightarrow & \multicolumn{2}{l}{(\text{bmi}\,(\text{muls-n}\,(\mathtt{64},\ x,\ y,\ \mathtt{32}))} \\
 & = & \textbf{if } (\text{nat-to-int}\,(x,\ \mathtt{32}) = \mathtt{0}) \lor (\text{nat-to-int}\,(y,\ \mathtt{32}) = \mathtt{0}) \\
 & & \textbf{then } \mathtt{0} \\
 & & \textbf{elseif } \text{negativep}\,(\text{nat-to-int}\,(x,\ \mathtt{32})) \\
 & & \textbf{then if } \text{negativep}\,(\text{nat-to-int}\,(y,\ \mathtt{32})) \ \textbf{ then } \mathtt{0} \\
 & & \qquad \textbf{else } \mathtt{1} \textbf{ endif} \\
 & & \textbf{elseif } \text{negativep}\,(\text{nat-to-int}\,(y,\ \mathtt{32})) \ \textbf{ then } \mathtt{1} \\
 & & \textbf{else } \mathtt{0} \textbf{ endif})
\end{array}
$$

THEOREM: muls_3232-bmi

$$
\begin{array}{lll}
\multicolumn{3}{l}{(\text{int-rangep}\,(\text{itimes}\,(\text{nat-to-int}\,(x,\ \mathtt{32}),\ \text{nat-to-int}\,(y,\ \mathtt{32})),\ \mathtt{32})} \\
\land & \multicolumn{2}{l}{\text{nat-rangep}\,(x,\ \mathtt{32})} \\
\land & \multicolumn{2}{l}{\text{nat-rangep}\,(y,\ \mathtt{32}))} \\
\rightarrow & \multicolumn{2}{l}{(\text{bmi}\,(\text{muls-n}\,(\mathtt{32},\ x,\ y,\ \mathtt{32}))} \\
 & = & \textbf{if } (\text{nat-to-int}\,(x,\ \mathtt{32}) = \mathtt{0}) \lor (\text{nat-to-int}\,(y,\ \mathtt{32}) = \mathtt{0}) \\
 & & \textbf{then } \mathtt{0} \\
 & & \textbf{elseif } \text{negativep}\,(\text{nat-to-int}\,(x,\ \mathtt{32})) \\
 & & \textbf{then if } \text{negativep}\,(\text{nat-to-int}\,(y,\ \mathtt{32})) \ \textbf{ then } \mathtt{0} \\
 & & \qquad \textbf{else } \mathtt{1} \textbf{ endif} \\
 & & \textbf{elseif } \text{negativep}\,(\text{nat-to-int}\,(y,\ \mathtt{32})) \ \textbf{ then } \mathtt{1} \\
 & & \textbf{else } \mathtt{0} \textbf{ endif})
\end{array}
$$

```
; the n-flag of asl.
```

THEOREM: asl-bmi

$(\text{nat-rangep}\,(x,\ n) \land \text{int-rangep}\,(\text{nat-to-int}\,(x,\ n),\ n - cnt))$

$\rightarrow$ (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-rangep $(x,\ n)$
$\rightarrow$ (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-rangep $(x,\ n) \land (n < size)$)
$\rightarrow$ (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-rangep $(x,\ n) \land$ nat-rangep $(y,\ n) \land (n \not\simeq 0)$)
$\rightarrow$ (bge (sub-v $(n,\ x,\ y)$, sub-n $(n,\ x,\ y)$)
  = **if** ilessp (nat-to-int $(y,\ n)$, nat-to-int $(x,\ n)$) **then** `0`
  else `1` **endif**)

`; blt of sub.`

THEOREM: sub-blt
(nat-rangep $(x,\ n) \land$ nat-rangep $(y,\ n) \land (n \not\simeq 0)$)
$\rightarrow$ (blt (sub-v $(n,\ x,\ y)$, sub-n $(n,\ x,\ y)$)
  = **if** ilessp (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
blt $(v,\ n)$ = b-not (bge $(v,\ n)$)

EVENT: Disable bge.

EVENT: Disable blt.

```
;                               BGT/BLE
```

THEOREM: sub-z-la1
(nat-rangep $(x,\ n)$ $\land$ nat-rangep $(y,\ n)$)
$\rightarrow$ (sub-z $(n,\ x,\ y)$
= **if** nat-to-int $(x,\ n)$ = nat-to-int $(y,\ n)$ **then** 1
else 0 **endif**)

EVENT: Disable sub-z.

```
; bgt of sub.
```

THEOREM: sub-bgt
(nat-rangep $(x,\ n)$ $\land$ nat-rangep $(y,\ n)$ $\land$ $(n \not\simeq 0)$)
$\rightarrow$ (bgt (sub-v $(n,\ x,\ y)$, sub-z $(n,\ x,\ y)$, sub-n $(n,\ x,\ y)$)
= **if** ilessp (nat-to-int $(x,\ n)$, nat-to-int $(y,\ n)$) **then** 1
else 0 **endif**)

```
; ble of sub.
```

THEOREM: sub-ble
(nat-rangep $(x,\ n)$ $\land$ nat-rangep $(y,\ n)$ $\land$ $(n \not\simeq 0)$)
$\rightarrow$ (ble (sub-v $(n,\ x,\ y)$, sub-z $(n,\ x,\ y)$, sub-n $(n,\ x,\ y)$)
= **if** ilessp (nat-to-int $(x,\ n)$, nat-to-int $(y,\ n)$) **then** 0
else 1 **endif**)

```
; bgt of move.
```

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

```
; ble of move.
```

THEOREM: move-ble
nat-rangep $(x,\ n)$
$\rightarrow$ (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 ilessp-crock1.

EVENT: Disable ilessp-crock2.

EVENT: Disable add-bmi-crock1.

EVENT: Disable add-bmi-crock2.

EVENT: Disable nat-to-int=.

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

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

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

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

THEOREM: stepn-rewriter
(mc-status (s) = `'running`) $\rightarrow$ (stepn $(s,\ 1 + n)$ = stepn (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
rom-addrp ($addr$, byte-write ($value$, x, mem), n) = rom-addrp ($addr$, mem, n)

THEOREM: write-mem-maintain-rom-addrp
rom-addrp ($addr$, write-mem ($value$, x, mem, m), n) = rom-addrp ($addr$, 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
ram-addrp ($addr$, byte-write ($value$, x, mem), n) = ram-addrp ($addr$, mem, n)

THEOREM: write-mem-maintain-ram-addrp
ram-addrp ($addr$, write-mem ($value$, x, mem, m), n) = ram-addrp ($addr$, mem, n)

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

DEFINITION:
mcode-addrp ($addr$, mem, lst)
= **if** listp (lst)
  then if car (lst) = `-1`
    then mcode-addrp (add (`32`, $addr$, `1`), mem, cdr (lst))
    else (pc-byte-read ($addr$, mem) = car (lst))
      $\wedge$ mcode-addrp (add (`32`, $addr$, `1`), mem, cdr (lst)) **endif**
  else t endif

THEOREM: add-non-numberp
($i \notin \mathbf{N}$) $\rightarrow$ (add (n, x, i) = head (x, n))

THEOREM: add-plus
add (n, x, add (n, y, z)) = add (n, x, $y + z$)

```
; four lemmas about pc-read-memp.
```

THEOREM: pc-read-memp-la0
(pc-read-memp ($addr$, mem, k) $\wedge$ ($k \not\simeq$ `0`)) $\rightarrow$ pc-byte-readp ($addr$, mem)

THEOREM: pc-read-memp-la1
(pc-read-memp ($addr$, mem, k) $\wedge$ ($j < k$))
$\rightarrow$ pc-byte-readp (add (`32`, $addr$, j), mem)

THEOREM: pc-read-memp-la2
(pc-read-memp ($addr$, mem, k) $\wedge$ ($j \leq k$)) $\rightarrow$ pc-read-memp ($addr$, mem, j)

THEOREM: pc-read-memp-la3
(pc-read-memp ($addr$, mem, k) $\wedge$ (($i + j$) $\leq k$))
$\rightarrow$ pc-read-memp (add (`32`, $addr$, i), mem, j)

THEOREM: write-memp-la0
(write-memp ($addr$, mem, n) $\wedge$ ($n \not\simeq$ `0`)) $\rightarrow$ byte-writep ($addr$, mem)

THEOREM: write-memp-la1
(write-memp ($addr$, mem, n) $\wedge$ ($m < n$)) $\rightarrow$ byte-writep (add (`32`, $addr$, m), mem)

THEOREM: write-memp-la2
(write-memp ($addr$, mem, n) $\wedge$ ($m \leq n$)) $\rightarrow$ write-memp ($addr$, mem, m)

THEOREM: write-memp-la3
(write-memp $(addr, mem, n) \wedge ((i + j) \leq n))$
$\rightarrow$ write-memp (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: index-n (x, y) = sub (32, y, x)

```
; ROM is pc-readable.
```

THEOREM: pc-byte-readp-rom0
(rom-addrp $(addr, mem, k) \wedge (k \not\simeq 0)) \rightarrow$ pc-byte-readp $(addr, mem)$

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

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

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

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

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

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

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

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

THEOREM: rom-addrp-la2
$(\text{rom-addrp}\,(addr,\ mem,\ k) \land ((j + \text{index-n}\,(x,\ addr)) \leq k))$
$\rightarrow \quad \text{rom-addrp}\,(x,\ mem,\ j)$

EVENT: Disable rom-addrp.

```
; ROM is readable.
```

THEOREM: byte-readp-rom0
$(\text{rom-addrp}\,(addr,\ mem,\ k) \land (k \not\simeq \mathtt{0})) \rightarrow \text{byte-readp}\,(addr,\ mem)$

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

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

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

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

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

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

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

```
; integer view.
```

THEOREM: read-memp-rom1-int
(rom-addrp $(addr,\ mem,\ n)$
 $\land$ ((nat-to-int $(x,\ 32) + j) < n$)
 $\land$ (nat-to-int $(x,\ 32) \in \mathbf{N}$))
$\rightarrow$ read-memp (add (32, $addr,\ x$), $mem,\ j$)

```
; RAM is writable.
```

THEOREM: byte-writep-ram0
(ram-addrp $(addr,\ mem,\ k) \land (k \not\simeq 0)$) $\rightarrow$ byte-writep $(addr,\ mem)$

THEOREM: byte-writep-ram1
(ram-addrp $(addr,\ mem,\ k) \land (j < k)$) $\rightarrow$ byte-writep (add (32, $addr,\ j$), mem)

THEOREM: byte-writep-ram2
(ram-addrp (add (32, $addr,\ i$), $mem,\ k$) $\land$ (index-n $(0,\ i) < k$))
$\rightarrow$ byte-writep $(addr,\ mem)$

THEOREM: byte-writep-ram3
(ram-addrp (add (32, $addr,\ i$), $mem,\ k$) $\land$ (index-n $(j,\ i) < k$))
$\rightarrow$ byte-writep (add (32, $addr,\ j$), mem)

THEOREM: write-memp-ram0
(ram-addrp $(addr,\ mem,\ k) \land (j \leq k)$) $\rightarrow$ write-memp $(addr,\ mem,\ j)$

THEOREM: write-memp-ram1
(ram-addrp $(addr,\ mem,\ k) \land ((i + j) \leq k)$)
$\rightarrow$ write-memp (add (32, $addr,\ i$), $mem,\ j$)

THEOREM: write-memp-ram1-int
(ram-addrp $(addr,\ mem,\ k)$
 $\land$ (nat-to-int $(i,\ 32) \in \mathbf{N}$)
 $\land$ ((nat-to-int $(i,\ 32) + j) \leq k$))
$\rightarrow$ write-memp (add (32, $addr,\ i$), $mem,\ j$)

THEOREM: write-memp-ram2
(ram-addrp (add (32, $addr,\ i$), $mem,\ k$) $\land$ ((index-n $(0,\ i) + j) \leq k$))
$\rightarrow$ write-memp $(addr,\ mem,\ j)$

THEOREM: write-memp-ram3
(ram-addrp (add (32, $addr,\ i$), $mem,\ k$) $\land$ ((index-n $(h,\ i) + j) \leq k$))
$\rightarrow$ write-memp (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
$(\text{ram-addrp}\,(addr,\ mem,\ n) \land (m \leq n)) \rightarrow \text{ram-addrp}\,(addr,\ mem,\ m)$

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

EVENT: Disable ram-addrp.

```
; RAM is readable.
```

THEOREM: byte-readp-ram0
$(\text{ram-addrp}\,(addr,\ mem,\ k) \land (k \not\simeq \mathtt{0})) \rightarrow \text{byte-readp}\,(addr,\ mem)$

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

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

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

THEOREM: read-memp-ram0
$(\text{ram-addrp}\,(addr,\ mem,\ k) \land (j \leq k)) \rightarrow \text{read-memp}\,(addr,\ mem,\ j)$

THEOREM: read-memp-ram1
$(\text{ram-addrp}\,(addr,\ mem,\ k) \land ((i + j) \leq k))$
$\rightarrow \quad \text{read-memp}\,(\text{add}\,(\mathtt{32},\ addr,\ i),\ mem,\ j)$

THEOREM: read-memp-ram1-int
$(\text{ram-addrp}\,(addr,\ mem,\ k)$
$\land \quad (\text{nat-to-int}\,(i,\ \mathtt{32}) \in \mathbf{N})$
$\land \quad ((\text{nat-to-int}\,(i,\ \mathtt{32}) + j) \leq k))$
$\rightarrow \quad \text{read-memp}\,(\text{add}\,(\mathtt{32},\ addr,\ i),\ mem,\ j)$

THEOREM: read-memp-ram2
$(\text{ram-addrp}\,(\text{add}\,(\mathtt{32},\ addr,\ i),\ mem,\ k) \land ((\text{index-n}\,(\mathtt{0},\ i) + j) \leq k))$
$\rightarrow \quad \text{read-memp}\,(addr,\ mem,\ j)$

THEOREM: read-memp-ram3
$(\text{ram-addrp}\,(\text{add}\,(\mathtt{32},\ addr,\ i),\ mem,\ k) \land ((\text{index-n}\,(h,\ i) + j) \leq k))$
$\rightarrow \quad \text{read-memp}\,(\text{add}\,(\mathtt{32},\ addr,\ h),\ mem,\ j)$

```
; please use the right induction.
```

DEFINITION:
mem-induct1 (i, $addr$, lst)
$=$ **if** $i \simeq \texttt{0}$ **then t**
else mem-induct1 ($i - 1$, add ($\texttt{32}$, $addr$, $\texttt{1}$), 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)) $\land$ ($x \neq \texttt{-1}$))
$\rightarrow$ (pc-byte-read ($addr$, mem) $= x$)

THEOREM: pc-byte-read-mcode1
(mcode-addrp ($addr$, mem, lst) $\land$ ($i <$ len (lst)) $\land$ (get-nth (i, lst) $\neq \texttt{-1}$))
$\rightarrow$ (pc-byte-read (add ($\texttt{32}$, $addr$, i), mem) $=$ get-nth (i, lst))

THEOREM: pc-byte-read-mcode2
(mcode-addrp (add ($\texttt{32}$, $addr$, i), mem, lst)
$\land$ (index-n ($\texttt{0}$, i) $<$ len (lst))
$\land$ (get-nth (index-n ($\texttt{0}$, i), lst) $\neq \texttt{-1}$))
$\rightarrow$ (pc-byte-read ($addr$, mem) $=$ get-nth (index-n ($\texttt{0}$, i), lst))

THEOREM: pc-byte-read-mcode3
(mcode-addrp (add ($\texttt{32}$, $addr$, i), mem, lst)
$\land$ (index-n (j, i) $<$ len (lst))
$\land$ (get-nth (index-n (j, i), lst) $\neq \texttt{-1}$))
$\rightarrow$ (pc-byte-read (add ($\texttt{32}$, $addr$, j), mem) $=$ get-nth (index-n (j, i), lst))

DEFINITION:
lst-numberp0 (lst, n)
$=$ **if** $n \simeq \texttt{0}$ **then** $\texttt{0}$
else (get-nth ($n - 1$, lst) $\in \mathbf{N}$) $\land$ lst-numberp0 (lst, $n - 1$) **endif**

DEFINITION:
read-lst0 (lst, n)
$=$ **if** $n \simeq \texttt{0}$ **then** $\texttt{0}$
else app (B, get-nth ($n - 1$, lst), read-lst0 (lst, $n - 1$)) **endif**

DEFINITION:
tail-lst (lst, n)
$=$ **if** $n \simeq \texttt{0}$ **then** lst
else tail-lst (cdr (lst), $n - 1$) **endif**

DEFINITION:
lst-numberp (m, lst, n) $=$ lst-numberp0 (tail-lst (lst, m), n)

DEFINITION: read-lst$(m,\ lst,\ n)$ = read-lst0 (tail-lst $(lst,\ m),\ n)$

THEOREM: get-nth-tail-lst
get-nth $(n,$ tail-lst $(x,\ m))$ = get-nth $(m + n,\ x)$

THEOREM: pc-read-mem-mcode0
(mcode-addrp $(addr,\ mem,\ lst) \wedge (j \leq$ len $(lst)) \wedge$ lst-numberp $(0,\ lst,\ j))$
$\rightarrow$ (pc-read-mem $(addr,\ mem,\ j)$ = read-lst $(0,\ lst,\ j))$

THEOREM: pc-read-mem-mcode1
(mcode-addrp $(addr,\ mem,\ lst)$
$\wedge$ $((i + j) \leq$ len $(lst))$
$\wedge$ lst-numberp $(i,\ lst,\ j))$
$\rightarrow$ (pc-read-mem (add $(32,\ addr,\ i),\ mem,\ j)$ = read-lst $(i,\ lst,\ j))$

EVENT: Disable read-lst.

EVENT: Disable lst-numberp.

THEOREM: pc-read-mem-mcode2
(mcode-addrp (add $(32,\ addr,\ i),\ mem,\ lst)$
$\wedge$ ((index-n $(0,\ i) + k) \leq$ len $(lst))$
$\wedge$ lst-numberp (index-n $(0,\ i),\ lst,\ k))$
$\rightarrow$ (pc-read-mem $(addr,\ mem,\ k)$ = read-lst (index-n $(0,\ i),\ lst,\ k))$

THEOREM: pc-read-mem-mcode3
(mcode-addrp (add $(32,\ addr,\ i),\ mem,\ lst)$
$\wedge$ ((index-n $(j,\ i) + k) \leq$ len $(lst))$
$\wedge$ lst-numberp (index-n $(j,\ i),\ lst,\ k))$
$\rightarrow$ (pc-read-mem (add $(32,\ addr,\ j),\ mem,\ k)$ = read-lst (index-n $(j,\ i),\ lst,\ k))$

```
; sometimes,  we obtain machine code by read-mem.
```

THEOREM: read->pc-read-mem
read-mem $(x,\ mem,\ k)$ = pc-read-mem $(x,\ mem,\ k)$

THEOREM: read-mem-mcode1-int
(mcode-addrp $(addr,\ mem,\ lst)$
$\wedge$ ((nat-to-int $(i,\ 32) + j) \leq$ len $(lst))$
$\wedge$ (nat-to-int $(i,\ 32) \in \mathbf{N})$
$\wedge$ lst-numberp (nat-to-int $(i,\ 32),\ lst,\ j))$
$\rightarrow$ (read-mem (add $(32,\ addr,\ i),\ mem,\ j)$
$=$ read-lst (nat-to-int $(i,\ 32),\ lst,\ j))$

THEOREM: read-mem-mcode2
(mcode-addrp (add (32, $addr$, i), mem, lst)
 $\wedge$ ((index-n (0, i) + k) $\leq$ len (lst))
 $\wedge$ lst-numberp (index-n (0, i), lst, k))
$\rightarrow$ (read-mem ($addr$, mem, k) = read-lst (index-n (0, i), lst, k))

THEOREM: read-mem-mcode3
(mcode-addrp (add (32, $addr$, i), mem, lst)
 $\wedge$ ((index-n (j, i) + k) $\leq$ len (lst))
 $\wedge$ lst-numberp (index-n (j, i), lst, k))
$\rightarrow$ (read-mem (add (32, $addr$, j), mem, k) = read-lst (index-n (j, i), lst, k))

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
(pc-read-memp (pc, mem, len (lst)) $\wedge$ byte-writep (x, mem))
$\rightarrow$ (mcode-addrp (pc, byte-write ($value$, x, mem), lst)
 = mcode-addrp (pc, mem, lst))

THEOREM: write-mem-mcode-addrp
(pc-read-memp (pc, mem, len (lst)) $\wedge$ write-memp (x, mem, n))
$\rightarrow$ (mcode-addrp (pc, write-mem ($value$, x, mem, n), lst)
 = mcode-addrp (pc, mem, lst))

```
;                DISJOINT0 AND DISJOINT
```

THEOREM: disjoint0-head
(disjoint0 (head (x, 32), m, y) = disjoint0 (x, m, y))
$\wedge$ (disjoint0 (x, m, head (y, 32)) = disjoint0 (x, m, y))

THEOREM: disjoint-head
(disjoint (head (x, 32), m, y, n) = disjoint (x, m, y, n))
$\wedge$ (disjoint (x, m, head (y, 32), n) = disjoint (x, m, y, n))

THEOREM: disjoint0-la0
(disjoint0 (a, m, b) $\wedge$ ($i < m$)) $\rightarrow$ ($\neg$ mod32-eq (add (32, a, i), b))

THEOREM: disjoint0-la1
(disjoint0 (a, m, b) $\wedge$ ($i < m$)) $\rightarrow$ disjoint0 (a, i, b)

THEOREM: disjoint0-la2
(disjoint0 (a, m, b) $\wedge$ (($i + j$) $\leq m$)) $\rightarrow$ disjoint0 (add (32, a, i), j, b)

THEOREM: disjoint-la0
$(\text{disjoint}\,(a,\ m,\ b,\ n) \land (j < n)) \rightarrow \text{disjoint0}\,(a,\ m,\ \text{add}\,(\mathtt{32},\ b,\ j))$

THEOREM: disjoint-la1
$(\text{disjoint}\,(a,\ m,\ b,\ n) \land (i < m) \land (j < n))$
$\rightarrow \quad (\neg\ \text{mod32-eq}\,(\text{add}\,(\mathtt{32},\ a,\ i),\ \text{add}\,(\mathtt{32},\ b,\ j)))$

THEOREM: disjoint-la2
$(\text{disjoint}\,(a,\ m,\ b,\ n) \land ((i + j) \leq m) \land (k < n))$
$\rightarrow \quad \text{disjoint0}\,(\text{add}\,(\mathtt{32},\ a,\ i),\ j,\ \text{add}\,(\mathtt{32},\ b,\ k))$

THEOREM: disjoint-la3
$(\text{disjoint}\,(a,\ m,\ b,\ n) \land ((i + j) \leq m) \land ((k + l) \leq n))$
$\rightarrow \quad \text{disjoint}\,(\text{add}\,(\mathtt{32},\ a,\ i),\ j,\ \text{add}\,(\mathtt{32},\ b,\ k),\ l)$

;

THEOREM: disjoint-0
$(\text{disjoint}\,(a,\ m,\ b,\ n) \land (j \leq m) \land (l \leq n)) \rightarrow \text{disjoint}\,(a,\ j,\ b,\ l)$

THEOREM: disjoint-1
$(\text{disjoint}\,(a,\ m,\ b,\ n) \land (j \leq m) \land ((k + l) \leq n))$
$\rightarrow \quad \text{disjoint}\,(a,\ j,\ \text{add}\,(\mathtt{32},\ b,\ k),\ l)$

THEOREM: disjoint-2
$(\text{disjoint}\,(a,\ m,\ b,\ n) \land ((i + j) \leq m) \land (l \leq n))$
$\rightarrow \quad \text{disjoint}\,(\text{add}\,(\mathtt{32},\ a,\ i),\ j,\ b,\ l)$

THEOREM: disjoint-3
$(\text{disjoint}\,(a,\ m,\ b,\ n) \land ((i + j) \leq m) \land ((k + l) \leq n))$
$\rightarrow \quad \text{disjoint}\,(\text{add}\,(\mathtt{32},\ a,\ i),\ j,\ \text{add}\,(\mathtt{32},\ b,\ k),\ l)$

;

THEOREM: disjoint-4
$(\text{disjoint}\,(\text{add}\,(\mathtt{32},\ a,\ i),\ m,\ b,\ n) \land ((j + \text{index-n}\,(\mathtt{0},\ i)) \leq m) \land (l \leq n))$
$\rightarrow \quad \text{disjoint}\,(a,\ j,\ b,\ l)$

THEOREM: disjoint-5
$(\text{disjoint}\,(\text{add}\,(\mathtt{32},\ a,\ i),\ m,\ b,\ n)$
$\land \quad ((j + \text{index-n}\,(\mathtt{0},\ i)) \leq m)$
$\land \quad ((k + l) \leq n))$
$\rightarrow \quad \text{disjoint}\,(a,\ j,\ \text{add}\,(\mathtt{32},\ b,\ k),\ l)$

THEOREM: disjoint-6
$(\text{disjoint}\,(\text{add}\,(\mathtt{32},\ a,\ i),\ m,\ b,\ n)$
$\land \quad ((j + \text{index-n}\,(i1,\ i)) \leq m)$
$\land \quad (l \leq n))$
$\rightarrow \quad \text{disjoint}\,(\text{add}\,(\mathtt{32},\ a,\ i1),\ j,\ b,\ l)$

THEOREM: disjoint-7
$(\text{disjoint}(\text{add}(\texttt{32}, a, i), m, b, n)$
$\land \quad ((j + \text{index-n}(i1, i)) \leq m)$
$\land \quad ((k + l) \leq n))$
$\rightarrow \quad \text{disjoint}(\text{add}(\texttt{32}, a, i1), j, \text{add}(\texttt{32}, b, k), l)$

;

THEOREM: disjoint-8
$(\text{disjoint}(a, m, \text{add}(\texttt{32}, b, k), n) \land (j \leq m) \land ((l + \text{index-n}(\texttt{0}, k)) \leq n))$
$\rightarrow \quad \text{disjoint}(a, j, b, l)$

THEOREM: disjoint-9
$(\text{disjoint}(a, m, \text{add}(\texttt{32}, b, k), n)$
$\land \quad ((i + j) \leq m)$
$\land \quad ((l + \text{index-n}(\texttt{0}, k)) \leq n))$
$\rightarrow \quad \text{disjoint}(\text{add}(\texttt{32}, a, i), j, b, l)$

THEOREM: disjoint-10
$(\text{disjoint}(a, m, \text{add}(\texttt{32}, b, k), n)$
$\land \quad (j \leq m)$
$\land \quad ((l + \text{index-n}(k1, k)) \leq n))$
$\rightarrow \quad \text{disjoint}(a, j, \text{add}(\texttt{32}, b, k1), l)$

THEOREM: disjoint-11
$(\text{disjoint}(a, m, \text{add}(\texttt{32}, b, k), n)$
$\land \quad ((i + j) \leq m)$
$\land \quad ((l + \text{index-n}(k1, k)) \leq n))$
$\rightarrow \quad \text{disjoint}(\text{add}(\texttt{32}, a, i), j, \text{add}(\texttt{32}, b, k1), l)$

;

THEOREM: disjoint-12
$(\text{disjoint}(\text{add}(\texttt{32}, a, i), m, \text{add}(\texttt{32}, b, k), n)$
$\land \quad ((j + \text{index-n}(\texttt{0}, i)) \leq m)$
$\land \quad ((l + \text{index-n}(\texttt{0}, k)) \leq n))$
$\rightarrow \quad \text{disjoint}(a, j, b, l)$

THEOREM: disjoint-13
$(\text{disjoint}(\text{add}(\texttt{32}, a, i), m, \text{add}(\texttt{32}, b, k), n)$
$\land \quad ((j + \text{index-n}(i1, i)) \leq m)$
$\land \quad ((l + \text{index-n}(\texttt{0}, k)) \leq n))$
$\rightarrow \quad \text{disjoint}(\text{add}(\texttt{32}, a, i1), j, b, l)$

THEOREM: disjoint-14
$(\text{disjoint}(\text{add}(32, a, i), m, \text{add}(32, b, k), n)$
$\land \quad ((j + \text{index-n}(0, i)) \leq m)$
$\land \quad ((l + \text{index-n}(k1, k)) \leq n))$
$\rightarrow \quad \text{disjoint}(a, j, \text{add}(32, b, k1), l)$

THEOREM: disjoint-15
$(\text{disjoint}(\text{add}(32, a, i), m, \text{add}(32, b, k), n)$
$\land \quad ((j + \text{index-n}(i1, i)) \leq m)$
$\land \quad ((l + \text{index-n}(k1, k)) \leq n))$
$\rightarrow \quad \text{disjoint}(\text{add}(32, a, i1), j, \text{add}(32, b, k1), l)$

```
; 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~ $(\text{disjoint}(a, m, b, n) \land (j \leq m) \land (l \leq n)) \rightarrow \text{disjoint}(b, l, a, j)$

THEOREM: disjoint-1~ $(\text{disjoint}(a, m, b, n) \land (j \leq m) \land ((k + l) \leq n))$
$\rightarrow \quad \text{disjoint}(\text{add}(32, b, k), l, a, j)$

THEOREM: disjoint-2~ $(\text{disjoint}(a, m, b, n) \land ((i + j) \leq m) \land (l \leq n))$
$\rightarrow \quad \text{disjoint}(b, l, \text{add}(32, a, i), j)$

THEOREM: disjoint-3~ $(\text{disjoint}(a, m, b, n) \land ((i + j) \leq m) \land ((k + l) \leq n))$
$\rightarrow \quad \text{disjoint}(\text{add}(32, b, k), l, \text{add}(32, a, i), j)$

```
;
```

THEOREM: disjoint-4~ $(\text{disjoint}(\text{add}(32, a, i), m, b, n) \land ((j + \text{index-n}(0, i)) \leq m) \land (l \leq n))$
$\rightarrow \quad \text{disjoint}(b, l, a, j)$

THEOREM: disjoint-5~ $(\text{disjoint}(\text{add}(32, a, i), m, b, n)$
$\land \quad ((j + \text{index-n}(0, i)) \leq m)$
$\land \quad ((k + l) \leq n))$
$\rightarrow \quad \text{disjoint}(\text{add}(32, b, k), l, a, j)$

THEOREM: disjoint-6~ $(\text{disjoint}(\text{add}(32, a, i), m, b, n)$
$\land \quad ((j + \text{index-n}(i1, i)) \leq m)$
$\land \quad (l \leq n))$
$\rightarrow \quad \text{disjoint}(b, l, \text{add}(32, a, i1), j)$

THEOREM: disjoint-7˜

$$(\text{disjoint}(\text{add}(32,\ a,\ i),\ m,\ b,\ n) \land ((j + \text{index-n}(i1,\ i)) \leq m) \land ((k + l) \leq n)) \rightarrow \text{disjoint}(\text{add}(32,\ b,\ k),\ l,\ \text{add}(32,\ a,\ i1),\ j)$$

;

THEOREM: disjoint-8˜

$$(\text{disjoint}(a,\ m,\ \text{add}(32,\ b,\ k),\ n) \land (j \leq m) \land ((l + \text{index-n}(0,\ k)) \leq n)) \rightarrow \text{disjoint}(b,\ l,\ a,\ j)$$

THEOREM: disjoint-9˜

$$(\text{disjoint}(a,\ m,\ \text{add}(32,\ b,\ k),\ n) \land ((i + j) \leq m) \land ((l + \text{index-n}(0,\ k)) \leq n)) \rightarrow \text{disjoint}(b,\ l,\ \text{add}(32,\ a,\ i),\ j)$$

THEOREM: disjoint-10˜

$$(\text{disjoint}(a,\ m,\ \text{add}(32,\ b,\ k),\ n) \land (j \leq m) \land ((l + \text{index-n}(k1,\ k)) \leq n)) \rightarrow \text{disjoint}(\text{add}(32,\ b,\ k1),\ l,\ a,\ j)$$

THEOREM: disjoint-11˜

$$(\text{disjoint}(a,\ m,\ \text{add}(32,\ b,\ k),\ n) \land ((i + j) \leq m) \land ((l + \text{index-n}(k1,\ k)) \leq n)) \rightarrow \text{disjoint}(\text{add}(32,\ b,\ k1),\ l,\ \text{add}(32,\ a,\ i),\ j)$$

;

THEOREM: disjoint-12˜

$$(\text{disjoint}(\text{add}(32,\ a,\ i),\ m,\ \text{add}(32,\ b,\ k),\ n) \land ((j + \text{index-n}(0,\ i)) \leq m) \land ((l + \text{index-n}(0,\ k)) \leq n)) \rightarrow \text{disjoint}(b,\ l,\ a,\ j)$$

THEOREM: disjoint-13˜

$$(\text{disjoint}(\text{add}(32,\ a,\ i),\ m,\ \text{add}(32,\ b,\ k),\ n) \land ((j + \text{index-n}(i1,\ i)) \leq m) \land ((l + \text{index-n}(0,\ k)) \leq n)) \rightarrow \text{disjoint}(b,\ l,\ \text{add}(32,\ a,\ i1),\ j)$$

THEOREM: disjoint-14˜

$$(\text{disjoint}(\text{add}(32,\ a,\ i),\ m,\ \text{add}(32,\ b,\ k),\ n) \land ((j + \text{index-n}(0,\ i)) \leq m) \land ((l + \text{index-n}(k1,\ k)) \leq n)) \rightarrow \text{disjoint}(\text{add}(32,\ b,\ k1),\ l,\ a,\ j)$$

THEOREM: disjoint-15˜

$$(\text{disjoint}(\text{add}(32,\ a,\ i),\ m,\ \text{add}(32,\ b,\ k),\ n) \land ((j + \text{index-n}(i1,\ i)) \leq m) \land ((l + \text{index-n}(k1,\ k)) \leq n)) \rightarrow \text{disjoint}(\text{add}(32,\ b,\ k1),\ l,\ \text{add}(32,\ a,\ i1),\ j)$$

```
; disjoint with asl.
```

THEOREM: times-plus-lessp-cancel
$((a \leq k) \land (b \leq k) \land (a < b))$
$\rightarrow \quad (((a + (i * k)) < (b + (j * k))) = (i \leq j))$

THEOREM: disjoint-2-asl
$(\text{disjoint}\,(a,\ m,\ b,\ n)$
$\land \quad (opsz = \exp\,(\mathtt{2},\ cnt))$
$\land \quad (\text{nat-to-int}\,(x,\ \mathtt{32}) < (m \div opsz))$
$\land \quad (\text{nat-to-int}\,(x,\ \mathtt{32}) \in \mathbf{N})$
$\land \quad (l \leq n))$
$\rightarrow \quad \text{disjoint}\,(\text{add}\,(\mathtt{32},\ a,\ \text{asl}\,(\mathtt{32},\ x,\ cnt)),\ opsz,\ b,\ l)$

THEOREM: disjoint-2~asl
$(\text{disjoint}\,(a,\ m,\ b,\ n)$
$\land \quad (opsz = \exp\,(\mathtt{2},\ cnt))$
$\land \quad (\text{nat-to-int}\,(x,\ \mathtt{32}) < (m \div opsz))$
$\land \quad (\text{nat-to-int}\,(x,\ \mathtt{32}) \in \mathbf{N})$
$\land \quad (l \leq n))$
$\rightarrow \quad \text{disjoint}\,(b,\ l,\ \text{add}\,(\mathtt{32},\ a,\ \text{asl}\,(\mathtt{32},\ x,\ cnt)),\ opsz)$

THEOREM: disjoint-3-asl
$(\text{disjoint}\,(a,\ m,\ b,\ n)$
$\land \quad (opsz = \exp\,(\mathtt{2},\ cnt))$
$\land \quad (\text{nat-to-int}\,(x,\ \mathtt{32}) < (m \div opsz))$
$\land \quad (\text{nat-to-int}\,(x,\ \mathtt{32}) \in \mathbf{N})$
$\land \quad ((k + l) \leq n))$
$\rightarrow \quad \text{disjoint}\,(\text{add}\,(\mathtt{32},\ a,\ \text{asl}\,(\mathtt{32},\ x,\ cnt)),\ opsz,\ \text{add}\,(\mathtt{32},\ b,\ k),\ l)$

THEOREM: disjoint-3~asl
$(\text{disjoint}\,(a,\ m,\ b,\ n)$
$\land \quad (opsz = \exp\,(\mathtt{2},\ cnt))$
$\land \quad (\text{nat-to-int}\,(x,\ \mathtt{32}) < (m \div opsz))$
$\land \quad (\text{nat-to-int}\,(x,\ \mathtt{32}) \in \mathbf{N})$
$\land \quad ((k + l) \leq n))$
$\rightarrow \quad \text{disjoint}\,(\text{add}\,(\mathtt{32},\ b,\ k),\ l,\ \text{add}\,(\mathtt{32},\ a,\ \text{asl}\,(\mathtt{32},\ x,\ cnt)),\ opsz)$

THEOREM: disjoint-9-asl
$(\text{disjoint}\,(a,\ m,\ \text{add}\,(\mathtt{32},\ b,\ k),\ n)$
$\land \quad (opsz = \exp\,(\mathtt{2},\ cnt))$
$\land \quad (\text{nat-to-int}\,(x,\ \mathtt{32}) < (m \div opsz))$
$\land \quad (\text{nat-to-int}\,(x,\ \mathtt{32}) \in \mathbf{N})$
$\land \quad ((l + \text{index-n}\,(\mathtt{0},\ k)) \leq n))$
$\rightarrow \quad \text{disjoint}\,(\text{add}\,(\mathtt{32},\ a,\ \text{asl}\,(\mathtt{32},\ x,\ cnt)),\ opsz,\ b,\ l)$

THEOREM: disjoint-9~asl
$(\text{disjoint}(a, m, \text{add}(32, b, k), n)$
$\land \quad (opsz = \exp(2, cnt))$
$\land \quad (\text{nat-to-int}(x, 32) < (m \div opsz))$
$\land \quad (\text{nat-to-int}(x, 32) \in \mathbf{N})$
$\land \quad ((l + \text{index-n}(0, k)) \leq n))$
$\rightarrow \quad \text{disjoint}(b, l, \text{add}(32, a, \text{asl}(32, x, cnt)), opsz)$

THEOREM: disjoint-11-asl
$(\text{disjoint}(a, m, \text{add}(32, b, k), n)$
$\land \quad (opsz = \exp(2, cnt))$
$\land \quad (\text{nat-to-int}(x, 32) < (m \div opsz))$
$\land \quad (\text{nat-to-int}(x, 32) \in \mathbf{N})$
$\land \quad ((l + \text{index-n}(k1, k)) \leq n))$
$\rightarrow \quad \text{disjoint}(\text{add}(32, a, \text{asl}(32, x, cnt)), opsz, \text{add}(32, b, k1), l)$

THEOREM: disjoint-11~asl
$(\text{disjoint}(a, m, \text{add}(32, b, k), n)$
$\land \quad (opsz = \exp(2, cnt))$
$\land \quad (\text{nat-to-int}(x, 32) < (m \div opsz))$
$\land \quad (\text{nat-to-int}(x, 32) \in \mathbf{N})$
$\land \quad ((l + \text{index-n}(k1, k)) \leq n))$
$\rightarrow \quad \text{disjoint}(\text{add}(32, b, k1), l, \text{add}(32, a, \text{asl}(32, x, cnt)), opsz)$

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
$(\text{disjoint0}(x, m, \text{add}(32, x, y)) = \text{disjoint0}(0, m, y))$
$\land \quad (\text{disjoint0}(\text{add}(32, x, y), m, x) = \text{disjoint0}(y, m, 0))$

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
$\text{disjoint}(x, m, x, n) = ((m \simeq 0) \lor (n \simeq 0))$

THEOREM: disjoint-deduction0
$\text{disjoint}(x, m, y, 0) \land \text{disjoint}(x, 0, y, n)$

THEOREM: disjoint-deduction1
$(\text{disjoint}(x, m, \text{add}(32, x, y), n) = \text{disjoint}(0, m, y, n))$
$\land \quad (\text{disjoint}(\text{add}(32, x, y), m, x, n) = \text{disjoint}(y, m, 0, n))$

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
$(\text{index-n}(x, 0) = \text{head}(x, 32)) \land (\text{index-n}(0, x) = \text{neg}(32, x))$

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

THEOREM: index-n-deduction0
$(\text{index-n}(x, \text{neg}(32, y)) = \text{index-n}(\text{add}(32, x, y), 0))$
$\land \quad (\text{index-n}(\text{neg}(32, x), y) = \text{index-n}(0, \text{add}(32, x, y)))$

THEOREM: index-n-deduction1
$(\text{index-n}(\text{add}(32, x, y), x) = \text{index-n}(y, 0))$
$\land \quad (\text{index-n}(y, \text{add}(32, y, x)) = \text{index-n}(0, x))$

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:
$\text{mem-lst}(opsz, a, mem, n, lst)$
$= \quad$ **if** $n \simeq 0$ **then** $lst \simeq$ **nil**
else $(\text{read-mem}(a, mem, opsz) = \text{car}(lst))$
$\land \quad \text{mem-lst}(opsz, \text{add}(\text{L}, a, opsz), mem, n - 1, \text{cdr}(lst))$ **endif**

DEFINITION:
$\text{mem-ilst}(opsz, a, mem, n, lst)$
$= \quad$ **if** $n \simeq 0$ **then** $lst \simeq$ **nil**
else $(\text{nat-to-int}(\text{read-mem}(a, mem, opsz), 8 * opsz) = \text{car}(lst))$
$\land \quad \text{mem-ilst}(opsz, \text{add}(\text{L}, a, opsz), mem, n - 1, \text{cdr}(lst))$ **endif**

```
; every element in lst is in "good" range.
```

THEOREM: mem-lst-nat-rangep
$(\text{mem-lst}\,(opsz,\ a,\ mem,\ n,\ lst) \land ((8 * opsz) \le k))$
$\rightarrow \quad \text{nat-rangep}\,(\text{get-nth}\,(i,\ lst),\ k)$

THEOREM: mem-ilst-int-rangep
$(\text{mem-ilst}\,(opsz,\ a,\ mem,\ n,\ lst) \land (oplen = (8 * opsz)))$
$\rightarrow \quad \text{int-rangep}\,(\text{get-nth}\,(i,\ lst),\ oplen)$

```
; every element in lst is a natural number.
```

THEOREM: mem-lst-numberp
$\text{mem-lst}\,(opsz,\ a,\ mem,\ n,\ lst) \rightarrow (\text{get-nth}\,(i,\ lst) \in \mathbf{N})$

```
; every element in lst is bounded by 2^(8*opsz).
```

THEOREM: mem-lst-lessp
$(\text{mem-lst}\,(opsz,\ a,\ mem,\ n,\ lst) \land (k = \exp\,(2,\ 8 * opsz)))$
$\rightarrow \quad (\text{get-nth}\,(i,\ lst) < k)$

```
; every element in lst is an integer.
```

THEOREM: mem-ilst-integerp
$\text{mem-ilst}\,(opsz,\ a,\ mem,\ n,\ lst) \rightarrow \text{integerp}\,(\text{get-nth}\,(i,\ lst))$

```
; a trivial, but useful, instantiation!
```

THEOREM: readm-mem-lst
$\text{mem-lst}\,(opsz,\ a,\ mem,\ n,\ \text{readm-mem}\,(opsz,\ a,\ mem,\ n))$

```
; head with mem-lst/mem-ilst.
```

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

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

THEOREM: read-mem-non-numberp
$(x \notin \mathbf{N}) \rightarrow (\text{read-mem}\,(x,\ mem,\ k) = \text{read-mem}\,(0,\ mem,\ k))$

THEOREM: mem-lst-non-numberp
$(x \notin \mathbf{N}) \rightarrow (\text{mem-lst}\,(opsz,\ x,\ mem,\ n,\ lst) = \text{mem-lst}\,(opsz,\ 0,\ mem,\ n,\ lst))$

THEOREM: mem-lst-same
$(\text{mem-lst}\,(opsz,\ x,\ mem,\ n,\ lst) \land (\text{nat-to-uint}\,(x) = \text{nat-to-uint}\,(y)))$
$\rightarrow \quad \text{mem-lst}\,(opsz,\ y,\ mem,\ n,\ lst)$

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
len (append (x, y)) $=$ (len (x) $+$ len (y))

THEOREM: get-vals-len
len (get-vals (m, lst, n)) $=$ fix (n)

THEOREM: bv-to-lst-len
$\text{len}(\text{bv-to-lst}(opsz,\ bv,\ n)) = \text{fix}(n)$

THEOREM: get-vals-proper-lstp
$\text{proper-lstp}(\text{get-vals}(m,\ lst,\ n))$

THEOREM: bv-to-lst-proper-lstp
$\text{proper-lstp}(\text{bv-to-lst}(opsz,\ bv,\ n))$

THEOREM: mcdr-listp-len
$\text{listp}(\text{mcdr}(n,\ lst)) = (n < \text{len}(lst))$

THEOREM: cdr-mcdr
$\text{cdr}(\text{mcdr}(n,\ lst)) = \text{mcdr}(1 + n,\ lst)$

THEOREM: mcar-mcar
$(m \leq n) \rightarrow (\text{mcar}(m,\ \text{mcar}(n,\ lst)) = \text{mcar}(m,\ lst))$

THEOREM: mcdr-mcdr
$\text{mcdr}(n,\ \text{mcdr}(m,\ lst)) = \text{mcdr}(m + n,\ lst)$

THEOREM: mcar-nth
$\text{get-nth}(n,\ \text{mcar}(m,\ lst))$
$= \quad \textbf{if}\ n < m\ \textbf{then}\ \text{get-nth}(n,\ lst)$
$\qquad \textbf{else}\ 0\ \textbf{endif}$

THEOREM: mcdr-nth
$\text{get-nth}(n,\ \text{mcdr}(m,\ lst)) = \text{get-nth}(m + n,\ lst)$

THEOREM: get-lst-cdr
$\text{get-lst}(opsz,\ m,\ \text{cdr}(lst),\ n) = \text{get-lst}(opsz,\ 1 + m,\ lst,\ n)$

THEOREM: get-lst-mcdr
$\text{get-lst}(opsz,\ m,\ \text{mcdr}(j,\ lst),\ n) = \text{get-lst}(opsz,\ m + j,\ lst,\ n)$

THEOREM: get-lst-mcar
$((m + n) \leq j)$
$\rightarrow \quad (\text{get-lst}(opsz,\ m,\ \text{mcar}(j,\ lst),\ n) = \text{get-lst}(opsz,\ m,\ lst,\ n))$

THEOREM: get-vals-cdr
$\text{get-vals}(m,\ \text{cdr}(lst),\ n) = \text{get-vals}(1 + m,\ lst,\ n)$

THEOREM: get-vals-mcdr
$\text{get-vals}(m,\ \text{mcdr}(j,\ lst),\ n) = \text{get-vals}(m + j,\ lst,\ n)$

THEOREM: get-vals-mcar
$((m + n) \leq j) \rightarrow (\text{get-vals}(m,\ \text{mcar}(j,\ lst),\ n) = \text{get-vals}(m,\ lst,\ n))$

THEOREM: get-nth-append
get-nth (i, append (x, y))
$=$ **if** $i <$ len (x) **then** get-nth (i, x)
else get-nth ($i -$ len (x), y) **endif**

THEOREM: get-vals-append
($n \leq$ len (x)) $\rightarrow$ (get-vals (0, append (x, y), n) $=$ get-vals (0, x, n))

THEOREM: put-vals-append
($n \leq$ len (x)) $\rightarrow$ (put-vals (append (x, y), m, lst, n) $=$ put-vals (x, m, lst, n))

```
; another induction hint for many theorems about memory.
```

DEFINITION:
mem-induct2 ($opsz$, $addr$, i, n, lst, j)
$=$ **if** $n \simeq 0$ **then t**
else mem-induct2 ($opsz$,
add (32, $addr$, $opsz$),
$i - 1$,
$n - 1$,
cdr (lst),
$j - opsz$) **endif**

THEOREM: read-mem-lst-la
(mem-lst ($opsz$, a, mem, n, lst)
$\wedge$ ((j **mod** $opsz$) $=$ 0)
$\wedge$ (($j \div opsz$) $< n$))
$\rightarrow$ (read-mem (add (32, a, j), mem, $opsz$) $=$ get-nth ($j \div opsz$, lst))

THEOREM: mem-lst-get-lst0
(mem-lst (1, a, mem, n, lst) $\wedge$ ($k \leq n$))
$\rightarrow$ (nat-to-uint (read-mem (a, mem, k)) $=$ get-lst (1, 0, lst, k))

THEOREM: mem-lst-get-lst
(mem-lst (1, a, mem, n, lst) $\wedge$ ((nat-to-uint (j) $+ k$) $\leq n$))
$\rightarrow$ (nat-to-uint (read-mem (add (32, a, j), mem, k))
$=$ get-lst (1, nat-to-uint (j), lst, k))

THEOREM: mem-lst-get-vals0
(mem-lst (1, a, mem, n, lst) $\wedge$ ($k \leq n$))
$\rightarrow$ (bv-to-lst (1, read-mem (a, mem, k), k) $=$ get-vals (0, lst, k))

THEOREM: mem-lst-get-vals
(mem-lst (1, a, mem, n, lst) $\wedge$ ((nat-to-uint (j) $+ k$) $\leq n$))
$\rightarrow$ (bv-to-lst (1, read-mem (add (32, a, j), mem, k), k)
$=$ get-vals (nat-to-uint (j), lst, k))

```
; read is equivalent to get-nth.
```

THEOREM: read-mem-lst0
$(\text{mem-lst}(\textit{opsz}, a, \textit{mem}, n, \textit{lst}) \land (n \not\simeq 0))$
$\rightarrow \quad (\text{nat-to-uint}(\text{read-mem}(a, \textit{mem}, \textit{opsz})) = \text{get-nth}(0, \textit{lst}))$

THEOREM: iread-mem-get0
$(\text{mem-ilst}(\textit{opsz}, a, \textit{mem}, n, \textit{lst}) \land (n \not\simeq 0) \land (\textit{oplen} = (8 * \textit{opsz})))$
$\rightarrow \quad (\text{nat-to-int}(\text{read-mem}(a, \textit{mem}, \textit{opsz}), \textit{oplen}) = \text{get-nth}(0, \textit{lst}))$

THEOREM: read-mem-lst
$(\text{mem-lst}(\textit{opsz}, a, \textit{mem}, n, \textit{lst})$
$\land \quad ((\text{nat-to-uint}(j) \ \mathbf{mod} \ \textit{opsz}) = 0)$
$\land \quad ((\text{nat-to-uint}(j) \div \textit{opsz}) < n))$
$\rightarrow \quad (\text{nat-to-uint}(\text{read-mem}(\text{add}(32, a, j), \textit{mem}, \textit{opsz}))$
$\quad = \quad \text{get-nth}(\text{nat-to-uint}(j) \div \textit{opsz}, \textit{lst}))$

THEOREM: read-mem-lst-int
$(\text{mem-lst}(\textit{opsz}, a, \textit{mem}, n, \textit{lst})$
$\land \quad ((\text{nat-to-int}(j, 32) \ \mathbf{mod} \ \textit{opsz}) = 0)$
$\land \quad ((\text{nat-to-int}(j, 32) \div \textit{opsz}) < n)$
$\land \quad (\text{nat-to-int}(j, 32) \in \mathbf{N}))$
$\rightarrow \quad (\text{nat-to-uint}(\text{read-mem}(\text{add}(32, a, j), \textit{mem}, \textit{opsz}))$
$\quad = \quad \text{get-nth}(\text{nat-to-int}(j, 32) \div \textit{opsz}, \textit{lst}))$

THEOREM: read-mem-ilst
$(\text{mem-ilst}(\textit{opsz}, a, \textit{mem}, n, \textit{lst})$
$\land \quad (\textit{oplen} = (8 * \textit{opsz}))$
$\land \quad ((\text{nat-to-uint}(j) \ \mathbf{mod} \ \textit{opsz}) = 0)$
$\land \quad ((\text{nat-to-uint}(j) \div \textit{opsz}) < n))$
$\rightarrow \quad (\text{nat-to-int}(\text{read-mem}(\text{add}(32, a, j), \textit{mem}, \textit{opsz}), \textit{oplen})$
$\quad = \quad \text{get-nth}(\text{nat-to-uint}(j) \div \textit{opsz}, \textit{lst}))$

THEOREM: read-mem-ilst-int
$(\text{mem-ilst}(\textit{opsz}, a, \textit{mem}, n, \textit{lst})$
$\land \quad (\textit{oplen} = (8 * \textit{opsz}))$
$\land \quad ((\text{nat-to-uint}(j) \ \mathbf{mod} \ \textit{opsz}) = 0)$
$\land \quad ((\text{nat-to-uint}(j) \div \textit{opsz}) < n)$
$\land \quad (\text{nat-to-int}(j, 32) \in \mathbf{N}))$
$\rightarrow \quad (\text{nat-to-int}(\text{read-mem}(\text{add}(32, a, j), \textit{mem}, \textit{opsz}), \textit{oplen})$
$\quad = \quad \text{get-nth}(\text{nat-to-uint}(j) \div \textit{opsz}, \textit{lst}))$

```
; write to some location else does not affect mem-lst.
```

THEOREM: write-else-mem-lst
$\text{disjoint}(a, \textit{opsz} * n, x, m)$
$\rightarrow \quad (\text{mem-lst}(\textit{opsz}, a, \text{write-mem}(\textit{value}, x, \textit{mem}, m), n, \textit{lst})$
$\quad = \quad \text{mem-lst}(\textit{opsz}, a, \textit{mem}, n, \textit{lst}))$

THEOREM: write-else-mem-ilst
disjoint $(a,\ opsz * n,\ x,\ m)$
$\rightarrow$ (mem-ilst $(opsz,\ a,$ write-mem $(value,\ x,\ mem,\ m),\ n,\ lst)$
  $=$ mem-ilst $(opsz,\ a,\ mem,\ n,\ lst))$

```
; some conditions that makes disjoint true.
```

THEOREM: disjoint0-leq
$(((a + m) \leq 4294967296) \wedge (b < a)) \rightarrow$ disjoint0 $(a,\ m,\ b)$

THEOREM: disjoint-leq
$(((a + m) \leq 4294967296) \wedge ((b + n) \leq a)) \rightarrow$ disjoint $(a,\ m,\ b,\ n)$

THEOREM: disjoint-leq1
$(((a + m) \leq 4294967296) \wedge ((b + n) \leq a)) \rightarrow$ disjoint $(b,\ n,\ a,\ m)$

THEOREM: plus-times-lessp
$(((x + (y * z)) < w) \wedge (z1 < z))$
$\rightarrow$ $((x + y + (y * z1)) < w)$

THEOREM: write-mem-lst-la
(mem-lst $(opsz,\ a,\ mem,\ n,\ lst)$
$\wedge$ uint-rangep $(opsz * n,\ 32)$
$\wedge$ $(v1 =$ head $(v,\ 8 * opsz))$
$\wedge$ $(j = (opsz * i))$
$\wedge$ $(i < n)$
$\wedge$ $(v \in \mathbf{N}))$
$\rightarrow$ mem-lst $(opsz,\ a,$ write-mem $(v,$ add $(32,\ a,\ j),\ mem,\ opsz),\ n,$ put-nth $(v1,\ i,\ lst))$

DEFINITION:
mem-induct3 $(v,\ v1,\ a,\ mem,\ i,\ lst,\ k)$
$=$ **if** $k \simeq 0$ **then t**
else mem-induct3 (tail $(v,\ 8)$,
tail $(v1,\ 8)$,
a,
byte-write $(v,$ add $(32,\ a,\ i + (k - 1)),\ mem)$,
i,
put-nth (head $(v,\ 8),\ i + (k - 1),\ lst)$,
$k - 1)$ **endif**

THEOREM: mem-lst-put-lst
(mem-lst $(1,\ a,\ mem,\ n,\ lst)$
$\wedge$ uint-rangep $(n,\ 32)$
$\wedge$ nat-rangep $(v,\ 8 * k)$
$\wedge$ (nat-to-uint $(j) = i)$

$\wedge$ ($v1$ = nat-to-uint (v))
$\wedge$ (($i + k$) $\leq n$)
$\wedge$ ($j \in \mathbf{N}$)
$\wedge$ ($v \in \mathbf{N}$))
$\rightarrow$ mem-lst (1, a, write-mem (v, add (32, a, j), mem, k), n, put-lst (1, $v1$, i, lst, k))

DEFINITION:
mem-induct4 (v, $vals$, a, mem, i, lst, k)
= **if** $k \simeq 0$ **then t**
else mem-induct4 (tail (v, 8),
get-vals (0, $vals$, $k - 1$),
a,
byte-write (v, add (32, a, $i + (k - 1)$), mem),
i,
put-nth (head (v, 8), $i + (k - 1)$, lst),
$k - 1$) **endif**

THEOREM: get-vals-0
((len (lst) = n) $\wedge$ proper-lstp (lst)) $\rightarrow$ (get-vals (0, lst, n) = lst)

THEOREM: mem-lst-put-vals
(mem-lst (1, a, mem, n, lst)
$\wedge$ uint-rangep (n, 32)
$\wedge$ nat-rangep (v, 8 $*$ k)
$\wedge$ (nat-to-uint (j) = i)
$\wedge$ ($vals$ = bv-to-lst (1, v, k))
$\wedge$ (($i + k$) $\leq n$)
$\wedge$ ($j \in \mathbf{N}$)
$\wedge$ ($v \in \mathbf{N}$))
$\rightarrow$ mem-lst (1, a, write-mem (v, add (32, a, j), mem, k), n, put-vals ($vals$, i, lst, k))

EVENT: Disable get-vals-0.

THEOREM: write-mem-lst
(mem-lst ($opsz$, a, mem, n, lst)
$\wedge$ uint-rangep ($opsz * n$, 32)
$\wedge$ nat-rangep (v, 8 $*$ $opsz$)
$\wedge$ (nat-to-uint (j) = ($opsz * i$))
$\wedge$ ($v1$ = nat-to-uint (v))
$\wedge$ ($i < n$)
$\wedge$ ($j \in \mathbf{N}$)
$\wedge$ ($v \in \mathbf{N}$))
$\rightarrow$ mem-lst ($opsz$, a, write-mem (v, add (32, a, j), mem, $opsz$), n, put-nth ($v1$, i, lst))

THEOREM: write-mem-ilst
(mem-ilst ($opsz$, a, mem, n, lst)
 $\wedge$ uint-rangep ($opsz * n$, 32)
 $\wedge$ (nat-to-uint (j) = ($opsz * i$))
 $\wedge$ nat-rangep (v, 8 $*$ $opsz$)
 $\wedge$ ($v1$ = nat-to-int (v, 8 $*$ $opsz$))
 $\wedge$ ($i < n$)
 $\wedge$ ($j \in \mathbf{N}$)
 $\wedge$ ($v \in \mathbf{N}$))
$\rightarrow$ mem-ilst ($opsz$,
 a,
 write-mem (v, add (32, a, j), mem, $opsz$),
 n,
 put-nth ($v1$, i, lst))

THEOREM: write-mem-lst0
(mem-lst ($opsz$, a, mem, n, lst)
 $\wedge$ uint-rangep ($opsz * n$, 32)
 $\wedge$ nat-rangep (v, 8 $*$ $opsz$)
 $\wedge$ ($v1$ = nat-to-uint (v))
 $\wedge$ ($n \not\simeq 0$)
 $\wedge$ ($v \in \mathbf{N}$))
$\rightarrow$ mem-lst ($opsz$, a, write-mem (v, a, mem, $opsz$), n, put-nth ($v1$, 0, lst))

THEOREM: write-mem-ilst0
(mem-ilst ($opsz$, a, mem, n, lst)
 $\wedge$ uint-rangep ($opsz * n$, 32)
 $\wedge$ nat-rangep (v, 8 $*$ $opsz$)
 $\wedge$ ($v1$ = nat-to-int (v, 8 $*$ $opsz$))
 $\wedge$ ($n \not\simeq 0$)
 $\wedge$ ($v \in \mathbf{N}$))
$\rightarrow$ mem-ilst ($opsz$, a, write-mem (v, a, mem, $opsz$), n, put-nth ($v1$, 0, lst))

THEOREM: write-mem-lst-int
(mem-lst ($opsz$, a, mem, n, lst)
 $\wedge$ uint-rangep ($opsz * n$, 32)
 $\wedge$ (($opsz * i$) = nat-to-int (j, 32))
 $\wedge$ (nat-to-int (j, 32) $\in \mathbf{N}$)
 $\wedge$ nat-rangep (v, 8 $*$ $opsz$)
 $\wedge$ ($v1$ = nat-to-uint (v))
 $\wedge$ ($i < n$)
 $\wedge$ ($opsz \not\simeq 0$)
 $\wedge$ ($v \in \mathbf{N}$))
$\rightarrow$ mem-lst ($opsz$, a, write-mem (v, add (32, a, j), mem, $opsz$), n, put-nth ($v1$, i, lst))

THEOREM: write-mem-ilst-int

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{mem-ilst}\,(\mathit{opsz},\ a,\ \mathit{mem},\ n,\ \mathit{lst})} \\
\land & \text{uint-rangep}\,(\mathit{opsz} * n,\ \mathtt{32}) \\
\land & ((\mathit{opsz} * i) = \text{nat-to-int}\,(j,\ \mathtt{32})) \\
\land & (\text{nat-to-int}\,(j,\ \mathtt{32}) \in \mathbf{N}) \\
\land & \text{nat-rangep}\,(v,\ \mathtt{8} * \mathit{opsz}) \\
\land & (\mathit{v1} = \text{nat-to-int}\,(v,\ \mathtt{8} * \mathit{opsz})) \\
\land & (i < n) \\
\land & (\mathit{opsz} \not\simeq \mathtt{0}) \\
\land & (v \in \mathbf{N})) \\
\rightarrow & \text{mem-ilst}\,(\mathit{opsz}, \\
& \qquad\qquad a, \\
& \qquad\qquad \text{write-mem}\,(v,\ \text{add}\,(\mathtt{32},\ a,\ j),\ \mathit{mem},\ \mathit{opsz}), \\
& \qquad\qquad n, \\
& \qquad\qquad \text{put-nth}\,(\mathit{v1},\ i,\ \mathit{lst}))
\end{array}
$$

```
;                      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

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{mem-lst}\,(\mathit{opsz},\ a,\ \mathit{mem},\ n,\ \mathit{lst})} \\
\land & (\mathit{opsz} = \exp\,(\mathtt{2},\ \mathit{cnt})) \\
\land & (\text{nat-to-int}\,(i,\ \mathtt{32}) < n) \\
\land & (\text{nat-to-int}\,(i,\ \mathtt{32}) \in \mathbf{N})) \\
\rightarrow & (\text{nat-to-uint}\,(\text{read-mem}\,(\text{add}\,(\mathtt{32},\ a,\ \text{asl}\,(\mathtt{32},\ i,\ \mathit{cnt})),\ \mathit{mem},\ \mathit{opsz})) \\
& \ = \ \text{get-nth}\,(\text{nat-to-int}\,(i,\ \mathtt{32}),\ \mathit{lst}))
\end{array}
$$

THEOREM: read-mem-ilst-asl

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{mem-ilst}\,(\mathit{opsz},\ a,\ \mathit{mem},\ n,\ \mathit{lst})} \\
\land & (\mathit{opsz} = \exp\,(\mathtt{2},\ \mathit{cnt})) \\
\land & (\mathit{oplen} = (\mathtt{8} * \mathit{opsz})) \\
\land & (\text{nat-to-int}\,(i,\ \mathtt{32}) < n) \\
\land & (\text{nat-to-int}\,(i,\ \mathtt{32}) \in \mathbf{N})) \\
\rightarrow & (\text{nat-to-int}\,(\text{read-mem}\,(\text{add}\,(\mathtt{32},\ a,\ \text{asl}\,(\mathtt{32},\ i,\ \mathit{cnt})),\ \mathit{mem},\ \mathit{opsz}),\ \mathit{oplen}) \\
& \ = \ \text{get-nth}\,(\text{nat-to-int}\,(i,\ \mathtt{32}),\ \mathit{lst}))
\end{array}
$$

THEOREM: write-mem-lst-asl

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{mem-lst}\,(\mathit{opsz},\ a,\ \mathit{mem},\ n,\ \mathit{lst})} \\
\land & \text{uint-rangep}\,(\mathit{opsz} * n,\ \mathtt{32}) \\
\land & (\mathit{opsz} = \exp\,(\mathtt{2},\ \mathit{cnt})) \\
\land & \text{nat-rangep}\,(v,\ \mathtt{8} * \mathit{opsz}) \\
\land & (j = \text{nat-to-int}\,(i,\ \mathtt{32})) \\
\land & (\mathit{v1} = \text{nat-to-uint}\,(v))
\end{array}
$$

$\wedge\quad (j < n)$
$\wedge\quad (j \in \mathbf{N})$
$\wedge\quad (v \in \mathbf{N}))$
$\rightarrow\quad$ mem-lst ($opsz$,
a,
write-mem (v, add ($\mathsf{32}$, a, asl ($\mathsf{32}$, i, cnt)), mem, $opsz$),
n,
put-nth ($v1$, j, lst))

THEOREM: write-mem-ilst-asl
(mem-ilst ($opsz$, a, mem, n, lst)
$\wedge\quad$ uint-rangep ($opsz * n$, $\mathsf{32}$)
$\wedge\quad (opsz = \exp(\mathsf{2}, cnt))$
$\wedge\quad$ nat-rangep (v, $\mathsf{8} * opsz$)
$\wedge\quad (v1 = \text{nat-to-int}(v, \mathsf{8} * opsz))$
$\wedge\quad (j = \text{nat-to-int}(i, \mathsf{32}))$
$\wedge\quad (j < n)$
$\wedge\quad (v \in \mathbf{N})$
$\wedge\quad (j \in \mathbf{N}))$
$\rightarrow\quad$ mem-ilst ($opsz$,
a,
write-mem (v, add ($\mathsf{32}$, a, asl ($\mathsf{32}$, i, cnt)), mem, $opsz$),
n,
put-nth ($v1$, j, lst))

THEOREM: times-lessp-cancel1
$((y * x) < (x + (z * x))) = ((x \not\simeq \mathsf{0}) \wedge (y \leq z))$

THEOREM: read-memp-ram1-asl
(ram-addrp ($addr$, mem, n)
$\wedge\quad (opsz = \exp(\mathsf{2}, cnt))$
$\wedge\quad (\text{nat-to-int}(x, \mathsf{32}) < (n \div opsz))$
$\wedge\quad (\text{nat-to-int}(x, \mathsf{32}) \in \mathbf{N}))$
$\rightarrow\quad$ read-memp (add ($\mathsf{32}$, $addr$, asl ($\mathsf{32}$, x, cnt)), mem, $opsz$)

THEOREM: read-memp-rom1-asl
(rom-addrp ($addr$, mem, n)
$\wedge\quad (opsz = \exp(\mathsf{2}, cnt))$
$\wedge\quad (\text{nat-to-int}(x, \mathsf{32}) < (n \div opsz))$
$\wedge\quad (\text{nat-to-int}(x, \mathsf{32}) \in \mathbf{N}))$
$\rightarrow\quad$ read-memp (add ($\mathsf{32}$, $addr$, asl ($\mathsf{32}$, x, cnt)), mem, $opsz$)

THEOREM: write-memp-ram1-asl
(ram-addrp ($addr$, mem, n)
$\wedge\quad (opsz = \exp(\mathsf{2}, cnt))$

$\wedge$ (nat-to-int (x, 32) $<$ ($n \div$ *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/WRITEM-MEM EVENTS
;
```

THEOREM: writem-mem-maintain-pc-byte-readp
pc-byte-readp (x, writem-mem (*opsz*, *vlst*, *addr*, *mem*)) = pc-byte-readp (x, *mem*)

THEOREM: writem-mem-maintain-pc-read-memp
pc-read-memp (x, writem-mem (*opsz*, *vlst*, *addr*, *mem*), n) = pc-read-memp (x, *mem*, n)

THEOREM: writem-mem-maintain-byte-readp
byte-readp (x, writem-mem (*opsz*, *vlst*, *addr*, *mem*)) = byte-readp (x, *mem*)

THEOREM: writem-mem-maintain-read-memp
read-memp (x, writem-mem (*opsz*, *vlst*, *addr*, *mem*), n) = read-memp (x, *mem*, n)

THEOREM: writem-mem-maintain-byte-writep
byte-writep (x, writem-mem (*opsz*, *vlst*, *addr*, *mem*)) = byte-writep (x, *mem*)

THEOREM: writem-mem-maintain-write-memp
write-memp (x, writem-mem (*opsz*, *vlst*, *addr*, *mem*), n) = write-memp (x, *mem*, n)

THEOREM: writem-mem-maintain-rom-addrp
rom-addrp (x, writem-mem (*opsz*, *vlst*, *addr*, *mem*), n) = rom-addrp (x, *mem*, n)

THEOREM: writem-mem-maintain-ram-addrp
ram-addrp (x, writem-mem (*opsz*, *vlst*, *addr*, *mem*), n) = ram-addrp (x, *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, writem-mem (*opsz*, *vlst*, *addr*, *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, $opsz * n$, $addr$, $opsz1 *$ len ($vlst$))
$\rightarrow$ (mem-ilst ($opsz$, a, writem-mem ($opsz1$, $vlst$, $addr$, mem), n, lst)
$=$ mem-ilst ($opsz$, a, mem, n, lst))

THEOREM: read-writem-mem
disjoint ($addr$, n, $addr1$, $opsz *$ len ($vlst$))
$\rightarrow$ (read-mem ($addr$, 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 ($opsz$, $addr$, write-mem ($value$, $addr1$, mem, k), n)
$=$ readm-mem ($opsz$, $addr$, mem, n))

DEFINITION:
modn-lst (n, lst)
$=$ **if** $lst \simeq$ **nil then nil**
else cons (head (car (lst), n), modn-lst (n, cdr (lst))) **endif**

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

THEOREM: readm-writem-mem
(uint-rangep ($opsz * n$, 32) $\wedge$ ($n =$ len ($vlst$)))
$\rightarrow$ (readm-mem ($opsz$, $addr$, writem-mem ($opsz$, $vlst$, $addr$, mem), n)
$=$ modn-lst (8 $* opsz$, $vlst$))

THEOREM: disjoint-leq-uint
(((nat-to-uint (a) $+ m$) $\leq$ 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) $+ m$) $\leq$ 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
mem-lst $(opsz, a, mem, m + n, lst)$
$=$ (mem-lst $(opsz, a, mem, m$, mcar $(m, lst))$
$\land$ mem-lst $(opsz$, add $(\texttt{32}, a, opsz * m), mem, n$, mcdr $(m, lst)))$

THEOREM: mem-lst-mcar
(mem-lst $(opsz, a, mem, n, lst) \land (k \leq n))$
$\rightarrow$ mem-lst $(opsz, a, mem, k$, mcar $(k, lst))$

THEOREM: mem-lst-mcar-1
mem-lst $(opsz, a, mem, n$, mcar $(n, lst))$
$\rightarrow$ mem-lst $(opsz, a, mem, n - 1$, mcar $(n - 1, lst))$

THEOREM: mem-lst-mcar-2
(mem-lst $(opsz, a, mem, n$, mcar $(n, lst)) \land (k \leq n))$
$\rightarrow$ mem-lst $(opsz, a, mem, n - k$, mcar $(n - k, lst))$

THEOREM: plus-difference
$(m + (n - m))$
$=$ **if** $n < m$ **then** fix (m)
else fix (n) **endif**

THEOREM: mem-lst-len
mem-lst $(opsz, a, mem, n, lst) \rightarrow (n \not< \text{len}(lst))$

THEOREM: mem-lst-mcdr
(mem-lst $(opsz, a, mem, n, lst) \land (i = (opsz * m)))$
$\rightarrow$ mem-lst $(opsz$, add $(\texttt{32}, a, i), mem, n - m$, mcdr $(m, lst))$

THEOREM: mem-lst-mcdr-0
mem-lst $(opsz, a, mem, n, lst)$
$\rightarrow$ mem-lst $(opsz$, add $(\texttt{32}, a, opsz), mem, n - 1$, mcdr $(\texttt{1}, lst))$

THEOREM: mem-lst-mcdr-uint
(mem-lst $(opsz, a, mem, n, lst)$
$\land$ (nat-to-uint $(i) = (opsz * m))$
$\land$ $(opsz \not\simeq \texttt{0}))$
$\rightarrow$ mem-lst $(opsz$, add $(\texttt{32}, a, i), mem, n - m$, mcdr $(m, lst))$

THEOREM: mem-lst-mcdr-1
mem-lst (1, add (32, x, i), *mem*, $n - j$, mcdr (k, *lst*))
$\rightarrow$ mem-lst (1, add (32, x, add (32, i, 1)), *mem*, $(n - 1) - j$, mcdr ($1 + k$, *lst*))

THEOREM: mem-lst-mcdr-uint-1
(mem-lst (*opsz*, add (32, a, b), *mem*, $n - j$, mcdr (k, *lst*))
$\land$ (nat-to-uint (i) = (*opsz* $*$ m))
$\land$ (*opsz* $\not\simeq$ 0))
$\rightarrow$ mem-lst (*opsz*,
add (32, a, add (32, b, i)),
mem,
$n - (m + j)$,
mcdr ($m + k$, *lst*))

EVENT: Disable mem-lst-len.

EVENT: Disable plus-difference.

```
;                    LSTCPY AND LSTMCPY
```

DEFINITION:
lstcpy (*i1*, *lst1*, *i2*, *lst2*, n)
= **if** $n \simeq 0$ **then** *lst1*
else lstcpy ($1 +$ *i1*,
put-nth (get-nth (*i2*, *lst2*), *i1*, *lst1*),
$1 +$ *i2*,
lst2,
$n - 1$) **endif**

THEOREM: lstcpy-0
lstcpy (*i1*, *lst1*, *i2*, *lst2*, 0) = *lst1*

THEOREM: lstcpy-lstcpy
((*j1* = (*h1* + *i1*)) $\land$ (*j2* = (*h1* + *i2*)))
$\rightarrow$ (lstcpy (*j1*, lstcpy (*i1*, *lst1*, *i2*, *lst2*, *h1*), *j2*, *lst2*, *h2*)
= lstcpy (*i1*, *lst1*, *i2*, *lst2*, *h1* + *h2*))

DEFINITION:
lstmcpy (*opsz*, *i1*, *lst1*, *i2*, *lst2*, n)
= **if** $n \simeq 0$ **then** *lst1*
else lstmcpy (*opsz*,
opsz + *i1*,
lstcpy (*i1*, *lst1*, *i2*, *lst2*, *opsz*),
opsz + *i2*,
lst2,
$n - 1$) **endif**

THEOREM: lstmcpy-cpy
$\text{lstmcpy}\,(h,\ i1,\ lst1,\ i2,\ lst2,\ n) = \text{lstcpy}\,(i1,\ lst1,\ i2,\ lst2,\ h * n)$

THEOREM: put-commutativity
$(j < i)$
$\rightarrow\quad (\text{put-nth}\,(v1,\ i,\ \text{put-nth}\,(v2,\ j,\ lst)) = \text{put-nth}\,(v2,\ j,\ \text{put-nth}\,(v1,\ i,\ lst)))$

THEOREM: lstcpy-put-nth
$(i < j1)$
$\rightarrow\quad (\text{put-nth}\,(v,\ i,\ \text{lstcpy}\,(j1,\ lst1,\ j2,\ lst2,\ n))$
$\quad\quad = \quad \text{lstcpy}\,(j1,\ \text{put-nth}\,(v,\ i,\ lst1),\ j2,\ lst2,\ n))$

```
; a variant for lstcpy.
```

DEFINITION:
$\text{lstcpy1}\,(i1,\ lst1,\ i2,\ lst2,\ n)$
$= \quad$ **if** $n \simeq 0$ **then** $lst1$
else $\text{lstcpy1}\,(i1,$
$\text{put-nth}\,(\text{get-nth}\,(i2 + (n - 1),\ lst2),$
$i1 + (n - 1),$
$lst1),$
$i2,$
$lst2,$
$n - 1)$ **endif**

THEOREM: lstcpy-add1
$\text{lstcpy}\,(i1,\ \text{put-nth}\,(\text{get-nth}\,(i2 + h,\ lst2),\ i1 + h,\ lst1),\ i2,\ lst2,\ h)$
$= \quad \text{lstcpy}\,(i1,\ lst1,\ i2,\ lst2,\ 1 + h)$

THEOREM: lstcpy-cpy1
$\text{lstcpy}\,(i1,\ lst1,\ i2,\ lst2,\ n) = \text{lstcpy1}\,(i1,\ lst1,\ i2,\ lst2,\ n)$

THEOREM: put-get-lst-is-cpy
$\text{mem-lst}\,(opsz,\ x,\ mem,\ n2,\ lst2)$
$\rightarrow\quad (\text{put-lst}\,(opsz,\ \text{get-lst}\,(opsz,\ i2,\ lst2,\ n),\ i1,\ lst1,\ n)$
$\quad\quad = \quad \text{lstcpy}\,(i1,\ lst1,\ i2,\ lst2,\ n))$

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.

```
;            MMOV1-LST, MMOV-LST1, MMOVN-LST AND MMOVN-LST1
```

DEFINITION:
mmov1-lst $(i,\ lst1,\ lst2,\ n)$
$=$ **if** $n \simeq 0$ **then** *lst1*
else mmov1-lst $(1 + i,\ \text{put-nth}\,(\text{get-nth}\,(i,\ lst2),\ i,\ lst1),\ lst2,\ n - 1)$ **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 $(\text{get-nth}\,(i - 1,\ lst2),\ i - 1,\ lst1),$
$lst2,$
$n - 1)$ **endif**

DEFINITION:
movn-lst $(n,\ lst1,\ lst2,\ i) = \text{put-vals}\,(\text{get-vals}\,(i,\ lst2,\ n),\ i,\ lst1,\ n)$

DEFINITION:
mmovn-lst $(h,\ lst1,\ lst2,\ i,\ nt)$
$=$ **if** $nt \simeq 0$ **then** *lst1*
else mmovn-lst $(h,\ \text{movn-lst}\,(h,\ lst1,\ lst2,\ i),\ lst2,\ h + i,\ nt - 1)$ **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
$(\text{mod32-eq}(\mathtt{0}, \text{neg}(\mathtt{32}, x)) = \text{mod32-eq}(x, \mathtt{0}))$
$\land \quad (\text{mod32-eq}(\text{neg}(\mathtt{32}, x), \mathtt{0}) = \text{mod32-eq}(x, \mathtt{0}))$

THEOREM: mod32-eq-deduction2
$(\text{mod32-eq}(x, \text{add}(\mathtt{32}, x, y)) = \text{mod32-eq}(\mathtt{0}, y))$
$\land \quad (\text{mod32-eq}(\text{add}(\mathtt{32}, x, y), x) = \text{mod32-eq}(y, \mathtt{0}))$

THEOREM: mod32-eq-deduction3
$\text{mod32-eq}(\text{add}(\mathtt{32}, x, y), \text{add}(\mathtt{32}, x, z)) = \text{mod32-eq}(y, z)$

EVENT: Disable mod32-eq.

```
; generate all the cases from between-ileq.
```

THEOREM: between-ileq-la
$(\text{integerp}(x) \land \text{integerp}(y) \land \text{integerp}(z))$
$\rightarrow \quad (\text{between-ileq}(x, y, z)$
$\quad = \quad \textbf{if } \text{ilessp}(z, x) \textbf{ then f}$
$\quad\quad \textbf{else } (x = y) \lor \text{between-ileq}(\text{iplus}(x, \mathtt{1}), y, z) \textbf{ endif})$

EVENT: Disable between-ileq.

```
; readm-mem, mem-lst, and mem-ilst is not changed if read-mem is not
; changed.
```

THEOREM: read-mem-plus
$\text{read-mem}(addr, mem, m + k)$
$= \quad \text{app}(\mathtt{8} * k, \text{read-mem}(\text{add}(\mathtt{32}, addr, m), mem, k), \text{read-mem}(addr, mem, m))$

THEOREM: stepn-readm-mem
$(\text{read-mem}(x, \text{mc-mem}(\text{stepn}(s, n)), opsz * k)$
$= \quad \text{read-mem}(x, \text{mc-mem}(s), opsz * k))$
$\rightarrow \quad (\text{readm-mem}(opsz, x, \text{mc-mem}(\text{stepn}(s, n)), k)$
$\quad = \quad \text{readm-mem}(opsz, x, \text{mc-mem}(s), k))$

THEOREM: stepn-mem-lst
$(\text{readm-mem}(opsz, x, \text{mc-mem}(\text{stepn}(s, n)), k) = \text{readm-mem}(opsz, x, \text{mc-mem}(s), k))$
$\rightarrow \quad (\text{mem-lst}(opsz, x, \text{mc-mem}(\text{stepn}(s, n)), k, lst)$
$\quad = \quad \text{mem-lst}(opsz, x, \text{mc-mem}(s), k, lst))$

THEOREM: stepn-mem-ilst
(readm-mem ($opsz$, x, mc-mem (stepn (s, n)), k) = readm-mem ($opsz$, x, mc-mem (s), k))
$\rightarrow$ (mem-ilst ($opsz$, x, mc-mem (stepn (s, n)), k, lst)
= mem-ilst ($opsz$, x, mc-mem (s), k, 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 oplen 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&eori-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
$$\begin{aligned}&(\mathrm{p}(x,\ \text{write-mem}(\mathit{value},\ y,\ \mathit{mem},\ m),\ k) = \mathrm{p}(x,\ \mathit{mem},\ k))\\ \land\quad &((\mathrm{p}(x,\ \mathit{mem},\ k) \land \text{write-memp}(y,\ \mathit{mem},\ m))\\ &\quad \rightarrow\quad (\mathrm{h}(x,\ \text{write-mem}(\mathit{value},\ y,\ \mathit{mem},\ m),\ k) = \mathrm{h}(x,\ \mathit{mem},\ k)))\end{aligned}$$

Simultaneously, we introduce the new function symbols p and h.

THEOREM: addr-index2-mem
$$\text{mc-mem}(\text{car}(\text{addr-index2}(\mathit{pc},\ \mathit{addr},\ \mathit{indexwd},\ s))) = \text{mc-mem}(s)$$

THEOREM: immediate-mem
$$\text{mc-mem}(\text{car}(\text{immediate}(\mathit{oplen},\ \mathit{pc},\ s))) = \text{mc-mem}(s)$$

THEOREM: effec-addr-mem
$$\text{mc-mem}(\text{car}(\text{effec-addr}(\mathit{oplen},\ \mathit{mode},\ \mathit{rn},\ s))) = \text{mc-mem}(s)$$

THEOREM: mc-instate-mem
$$\text{mc-mem}(\text{car}(\text{mc-instate}(\mathit{oplen},\ \mathit{ins},\ s))) = \text{mc-mem}(s)$$

THEOREM: mapping-h
$$\begin{aligned}&\mathrm{p}(x,\ \text{mc-mem}(\text{car}(\mathit{s\&addr})),\ k)\\ \rightarrow\quad &(\mathrm{h}(x,\ \text{mc-mem}(\text{mapping}(\mathit{oplen},\ \mathit{v\&cvznx},\ \mathit{s\&addr})),\ k)\\ &\quad =\quad \mathrm{h}(x,\ \text{mc-mem}(\text{car}(\mathit{s\&addr})),\ k))\end{aligned}$$

DEFINITION: $\mathrm{t3}(x,\ y,\ z) = \mathbf{t}$

```
; add-group.
```

THEOREM: add-group-h
$$\begin{aligned}&\mathrm{p}(x,\ \text{mc-mem}(s),\ k)\\ \rightarrow\quad &(\mathrm{h}(x,\ \text{mc-mem}(\text{add-group}(\mathit{opmode},\ \mathit{ins},\ s)),\ k) = \mathrm{h}(x,\ \text{mc-mem}(s),\ k))\end{aligned}$$

```
; sub-group.
```

THEOREM: sub-group-h
$p(x, \text{mc-mem}(s), k)$
$\rightarrow \quad (h(x, \text{mc-mem}(\text{sub-group}(opmode, ins, s)), k) = h(x, \text{mc-mem}(s), k))$

```
; and-group.
```

THEOREM: and-group-h
$p(x, \text{mc-mem}(s), k)$
$\rightarrow \quad (h(x, \text{mc-mem}(\text{and-group}(oplen, ins, s)), k) = h(x, \text{mc-mem}(s), k))$

```
; or-group.
```

THEOREM: or-group-h
$p(x, \text{mc-mem}(s), k)$
$\rightarrow \quad (h(x, \text{mc-mem}(\text{or-group}(oplen, ins, s)), k) = h(x, \text{mc-mem}(s), k))$

```
; scc-group.
```

THEOREM: scc-group-h
$p(x, \text{mc-mem}(s), k) \rightarrow (h(x, \text{mc-mem}(\text{scc-group}(ins, s)), k) = h(x, \text{mc-mem}(s), k))$

```
; cmp-group.
```

THEOREM: cmp-group-h
$p(x, \text{mc-mem}(s), k)$
$\rightarrow \quad (h(x, \text{mc-mem}(\text{cmp-group}(oplen, ins, s)), k) = h(x, \text{mc-mem}(s), k))$

```
; bcc-group.
```

THEOREM: bcc-group-h
$p(x, \text{mc-mem}(s), k)$
$\rightarrow \quad (h(x, \text{mc-mem}(\text{bcc-group}(disp, ins, s)), k) = h(x, \text{mc-mem}(s), k))$

```
; bit-group.
```

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

THEOREM: movep-write-h
$(p(x, mem, k) \land \text{movep-writep}(addr, mem, n))$
$\rightarrow \quad (h(x, \text{movep-write}(value, addr, mem, n), k) = h(x, mem, k))$

THEOREM: bit-group-h
$p(x, \text{mc-mem}(s), k) \rightarrow (h(x, \text{mc-mem}(\text{bit-group}(ins, s)), k) = h(x, \text{mc-mem}(s), k))$

```
; move-group.
```

THEOREM: move-h
$\mathrm{p}(x, \text{mc-mem}(s), k)$
$\rightarrow \quad (\mathrm{h}(x, \text{mc-mem}(\text{move-ins}(oplen, ins, s)), k) = \mathrm{h}(x, \text{mc-mem}(s), k))$

THEOREM: move-group-h
$\mathrm{p}(x, \text{mc-mem}(s), k)$
$\rightarrow \quad (\mathrm{h}(x, \text{mc-mem}(\text{move-group}(oplen, ins, s)), k) = \mathrm{h}(x, \text{mc-mem}(s), k))$

`; s&r-group.`

THEOREM: s&r-group-h
$\mathrm{p}(x, \text{mc-mem}(s), k) \rightarrow (\mathrm{h}(x, \text{mc-mem}(\text{s\&r-group}(ins, s)), k) = \mathrm{h}(x, \text{mc-mem}(s), k))$

`; misc-group.`

THEOREM: writem-mem-h
$(\mathrm{p}(x, mem, k) \wedge \text{write-memp}(addr, mem, opsz * \text{len}(vlst)))$
$\rightarrow \quad (\mathrm{h}(x, \text{writem-mem}(opsz, vlst, addr, mem), k) = \mathrm{h}(x, mem, k))$

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
$\mathrm{p}(x, \text{mc-mem}(s), k)$
$\rightarrow \quad (\mathrm{h}(x, \text{mc-mem}(\text{misc-group}(ins, s)), k) = \mathrm{h}(x, \text{mc-mem}(s), k))$

THEOREM: stepi-h
$\mathrm{p}(x, \text{mc-mem}(s), k) \rightarrow (\mathrm{h}(x, \text{mc-mem}(\text{stepi}(s)), k) = \mathrm{h}(x, \text{mc-mem}(s), k))$

THEOREM: stepi-p
$\text{t3}(x, \text{mc-mem}(s), k) \rightarrow (\mathrm{p}(x, \text{mc-mem}(\text{stepi}(s)), k) = \mathrm{p}(x, \text{mc-mem}(s), k))$

THEOREM: stepn-h
$\mathrm{p}(x, \text{mc-mem}(s), k) \rightarrow (\mathrm{h}(x, \text{mc-mem}(\text{stepn}(s, n)), k) = \mathrm{h}(x, \text{mc-mem}(s), k))$

`; the instantiations.`

THEOREM: stepn-pc-read-memp
$\text{t3}(x, \text{mc-mem}(s), k)$
$\rightarrow \quad (\text{pc-read-memp}(x, \text{mc-mem}(\text{stepn}(s, n)), k) = \text{pc-read-memp}(x, \text{mc-mem}(s), k))$

THEOREM: stepn-read-memp
$\text{t3}(x, \text{mc-mem}(s), k)$
$\rightarrow \quad (\text{read-memp}(x, \text{mc-mem}(\text{stepn}(s, n)), k) = \text{read-memp}(x, \text{mc-mem}(s), k))$

THEOREM: stepn-write-memp
t3 $(x,\ \text{mc-mem}(s),\ k)$
$\rightarrow\quad (\text{write-memp}(x,\ \text{mc-mem}(\text{stepn}(s,\ n)),\ k) = \text{write-memp}(x,\ \text{mc-mem}(s),\ k))$

```
; after the execution of n arbitrary instructions, ROM is still ROM.
```

THEOREM: stepn-rom-addrp
$\text{rom-addrp}(x,\ \text{mc-mem}(\text{stepn}(s,\ n)),\ k) = \text{rom-addrp}(x,\ \text{mc-mem}(s),\ k)$

```
; after the execution of n arbitrary instructions, RAM is still RAM.
```

THEOREM: stepn-ram-addrp
$\text{ram-addrp}(x,\ \text{mc-mem}(\text{stepn}(s,\ n)),\ k) = \text{ram-addrp}(x,\ \text{mc-mem}(s),\ k)$

```
; 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
$\text{rom-addrp}(x,\ \text{mc-mem}(s),\ k)$
$\rightarrow\quad (\text{pc-read-mem}(x,\ \text{mc-mem}(\text{stepn}(s,\ n)),\ k) = \text{pc-read-mem}(x,\ \text{mc-mem}(s),\ k))$

THEOREM: stepn-read-mem
$\text{rom-addrp}(x,\ \text{mc-mem}(s),\ k)$
$\rightarrow\quad (\text{read-mem}(x,\ \text{mc-mem}(\text{stepn}(s,\ n)),\ k) = \text{read-mem}(x,\ \text{mc-mem}(s),\ k))$

```
; after the execution of n arbitrary instructions, the program segment
; maintains the same.  Since we always require programs in ROM.
```

THEOREM: stepn-mcode-addrp
$\text{rom-addrp}(x,\ \text{mc-mem}(s),\ \text{len}(lst))$
$\rightarrow\quad (\text{mcode-addrp}(x,\ \text{mc-mem}(\text{stepn}(s,\ n)),\ lst)$
$\quad\quad = \quad \text{mcode-addrp}(x,\ \text{mc-mem}(s),\ lst))$

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: uread-mem $(x,\ mem,\ n)$ = nat-to-uint (read-mem $(x,\ mem,\ n)$)

DEFINITION:
uread-dn $(oplen,\ dn,\ s)$ = nat-to-uint (read-dn $(oplen,\ dn,\ s)$)

DEFINITION:
uread-an $(oplen,\ an,\ s)$ = nat-to-uint (read-an $(oplen,\ an,\ s)$)

```
; look up as signed integers.
```

DEFINITION:
iread-mem $(x,\ mem,\ n)$ = nat-to-int (read-mem $(x,\ mem,\ n)$, $8 * n$)

DEFINITION:
iread-dn $(oplen,\ dn,\ s)$ = nat-to-int (read-dn $(oplen,\ dn,\ s)$, $oplen$)

DEFINITION:
iread-an $(oplen,\ an,\ s)$ = nat-to-int (read-an $(oplen,\ an,\ s)$, $oplen$)

```
; the return address of subroutine call.
```

DEFINITION:
rts-addr (s) = read-mem (read-an (32, 7, s), mc-mem (s), 4)

DEFINITION:
linked-rts-addr (s) = read-mem (add (32, read-an (32, 6, s), 4), mc-mem (s), 4)

```
; the saved A6 on the stack.
```

DEFINITION:
linked-a6 (s) = read-mem (read-an (32, 6, s), 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:
movem-saved $(s,\ opsz,\ i,\ n)$
= readm-mem ($opsz$, sub (32, i, read-an (32, 6, s)), mc-mem (s), n)

```
; when only one register is saved,  we use MOVE instead of MOVEM.
```

DEFINITION:
rn-saved (s) = read-mem (sub (32, 4, read-an (32, 6, s)), 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** $\mathit{lst} \simeq$ **nil** **then** lst
else cons (nat-to-int (car $(\mathit{lst}), n)$, lst-int $(n$, cdr $(\mathit{lst})))$ **endif**

THEOREM: get-lst-int
get-nth $(i$, lst-int $(n, \mathit{lst}))$ $=$ nat-to-int (get-nth $(i, \mathit{lst}), n)$

THEOREM: put-lst-int
put-nth (nat-to-int $(\mathit{value}, n), i$, lst-int $(n, \mathit{lst}))$
$=$ lst-int $(n$, put-nth $(\mathit{value}, i, \mathit{lst}))$

DEFINITION:
lst-integerp (lst)
$=$ **if** $\mathit{lst} \simeq$ **nil** **then t**
else integerp (car $(\mathit{lst})) \wedge$ lst-integerp (cdr $(\mathit{lst}))$ **endif**

THEOREM: get-lst-integerp
lst-integerp $(\mathit{lst}) \rightarrow$ integerp (get-nth $(n, \mathit{lst}))$

THEOREM: mem-lst-integerp
(mem-lst $(\mathit{opsz}, a, \mathit{mem}, n, \mathit{lst}) \wedge (\mathit{oplen} = (8 * \mathit{opsz})))$
$\rightarrow$ lst-integerp (lst-int $(\mathit{oplen}, \mathit{lst}))$

THEOREM: mem-ilst-lst-integerp
mem-ilst $(\mathit{opsz}, a, \mathit{mem}, n, \mathit{lst}) \rightarrow$ 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*
equal* (read-rn ($oplen$, rn, $rfile$), x) $\rightarrow$ (read-rn ($oplen$, 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:
d2-7a2-5p (rn)
$= \quad ((rn \not\simeq 0)$
$\quad \land \quad (rn \neq 1)$
$\quad \land \quad (rn \neq 8)$
$\quad \land \quad (rn \neq 9)$
$\quad \land \quad (rn \neq 14)$
$\quad \land \quad (rn \neq 15))$

```
; d2 will be used for some purpose.
```

DEFINITION: d3-7a2-5p (rn) = (d2-7a2-5p (rn) $\land$ $(rn \neq 2)$)

```
; d2 and d3 will be used for some purpose.
```

DEFINITION:
d4-7a2-5p (rn) = (d2-7a2-5p (rn) $\land$ $(rn \neq 2) \land (rn \neq 3)$)

```
; d2, d3, and d4 will be used for some purpose.
```

DEFINITION:
d5-7a2-5p (rn)
= (d2-7a2-5p (rn) $\land$ $(rn \neq 2) \land (rn \neq 3) \land (rn \neq 4)$)

```
; a2 will be used for some purpose.
```

DEFINITION: d2-7a3-5p (rn) = (d2-7a2-5p (rn) $\land$ $(rn \neq 10)$)

DEFINITION:
d4-7a4-5p (rn) = (d4-7a2-5p (rn) $\land$ $(rn \neq 10) \land (rn \neq 11)$)

DEFINITION:
d4-7a5p (rn)
= (d4-7a2-5p (rn) $\land$ $(rn \neq 10) \land (rn \neq 11) \land (rn \neq 12)$)

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
$((((i + j) \div 2) < j) = (i < j))$
$\wedge \quad ((((j + i) \div 2) < j) = (i < j))$

```
; swap the ith and jth 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** $n = \texttt{0}$ **then f**
else memchr1 $(\texttt{0},\ n,\ lst,\ ch)$ **endif**

```
; memcmp.
```

DEFINITION:
memcmp1 $(i,\ n,\ lst1,\ lst2)$
$=$ **if** get-nth $(i,\ lst1)$ $=$ get-nth $(i,\ lst2)$
then if $(n - 1) = \texttt{0}$ **then** $\texttt{0}$
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** $n = \texttt{0}$ **then** $\texttt{0}$
else memcmp1 $(\texttt{0},\ n,\ lst1,\ lst2)$ **endif**

```
; memset.
```

DEFINITION:
memset1 $(i,\ n,\ lst,\ ch)$
$=$ **if** $(n - 1) = \texttt{0}$ **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** $n = \texttt{0}$ **then** lst
else memset1 $(\texttt{0},\ n,\ lst,\ ch)$ **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** $n = \mathtt{0}$ **then** $lst1$
elseif get-nth ($\mathtt{0}$, $lst1$) = NULL **then** strcpy2 ($\mathtt{0}$, $lst1$, $\mathtt{0}$, n, $lst2$)
else strcpy2 (strlen ($\mathtt{1}$, $n1$, $lst1$), $lst1$, $\mathtt{0}$, n, $lst2$) **endif**

```
; strncmp.
```

DEFINITION:
strncmp1 (i, n, $lst1$, $lst2$)
= **if** get-nth (i, $lst1$) = get-nth (i, $lst2$)
then if get-nth (i, $lst1$) = $\mathtt{0}$ **then** $\mathtt{0}$
elseif $(n - 1) = \mathtt{0}$ **then** $\mathtt{0}$
else strncmp1 ($1 + i$, $n - 1$, $lst1$, $lst2$) **endif**
else idifference (get-nth (i, $lst1$), get-nth (i, $lst2$)) **endif**

DEFINITION:
strncmp (n, $lst1$, $lst2$)
= **if** $n \simeq \mathtt{0}$ **then** $\mathtt{0}$
else strncmp1 ($\mathtt{0}$, n, $lst1$, $lst2$) **endif**

```
; strncpy.
```

DEFINITION:
zero-list1 (i, n, lst)
= **if** $(n - 1) = \mathtt{0}$ **then** lst
else zero-list1 ($1 + i$, $n - 1$, put-nth ($\mathtt{0}$, i, lst)) **endif**

DEFINITION:
zero-list (i, n, lst) = zero-list1 ($1 + i$, n, put-nth ($\mathtt{0}$, i, lst))

DEFINITION:
strncpy1 (i, n, $lst1$, $lst2$)
= **if** get-nth (i, $lst2$) = $\mathtt{0}$ **then** zero-list (i, n, $lst1$)
elseif $(n - 1) = \mathtt{0}$ **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** $n \simeq \mathtt{0}$ **then** $lst1$
else strncpy1 ($\mathtt{0}$, 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 $(i,\ lst) = \texttt{0}$ **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 $(\texttt{0},\ n2,\ lst2,$ 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 $(i,\ lst) = \texttt{0}$ **then** fix (i)
else strrchr $(1 + i,\ n,\ lst,\ ch,$ fix $(i))$ **endif**
elseif get-nth $(i,\ lst) = \texttt{0}$ **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 $(i,\ lst) = \texttt{0}$ **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
$(n \not< \text{strchr1}(i,\ n,\ lst,\ ch))$
$\land \quad (\text{strchr1}(i,\ n,\ lst,\ ch) \to (\text{strchr1}(i,\ n,\ lst,\ ch) \not< i))$

THEOREM: strchr1-false-0
$\neg\ \text{strchr1}(i,\ n,\ lst,\ \texttt{0})$

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:
strstr1 (i, $n1$, $lst1$, $n2$, $lst2$, len)
= **if** $i < n1$
then let j **be** strchr1 (i, $n1$, $lst1$, get-nth (0, $lst2$))
in
if $j \in \mathbf{N}$
then if strncmp (len, mcdr ($1 + j$, $lst1$), mcdr (1, $lst2$))
= 0 **then** j
else strstr1 ($1 + j$, $n1$, $lst1$, $n2$, $lst2$, 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, $n1$, $lst1$, $n2$, $lst2$, strlen (0, $n2 - 1$, mcdr (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 $(n - 1) = \texttt{0}$ **then f**
else memchr* (add ($\texttt{32}$, $i*$, $\texttt{1}$), $1 + i$, $n - 1$, 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) $= \texttt{0}$ **then f**
else strchr* (add ($\texttt{32}$, $i*$, $\texttt{1}$), $1 + i$, n, lst, ch) **endif**
else f endif

DEFINITION:
strchr1* ($i*$, i, n, lst, ch)
$=$ **if** $i < n$
then if get-nth (i, lst) $= \texttt{0}$ **then f**
elseif get-nth (i, lst) $=$ ch **then** $i*$
else strchr1* (add ($\texttt{32}$, $i*$, $\texttt{1}$), $1 + i$, n, lst, ch) **endif**
else f endif

DEFINITION:
strcspn* ($i1*$, $i1$, $n1$, $lst1$, $n2$, $lst2$)
$=$ **if** $i1 < n1$
then if strchr ($\texttt{0}$, $n2$, $lst2$, get-nth ($i1$, $lst1$)) **then** $i1*$
else strcspn* (add ($\texttt{32}$, $i1*$, $\texttt{1}$), $1 + i1$, $n1$, $lst1$, $n2$, $lst2$) **endif**
else f endif

DEFINITION:
strrchr* ($i*$, i, n, lst, ch, $j*$)
$=$ **if** $i < n$
then if get-nth (i, lst) $=$ ch
then if get-nth (i, lst) $= \texttt{0}$ **then** $i*$
else strrchr* (add ($\texttt{32}$, $i*$, $\texttt{1}$), $1 + i$, n, lst, ch, $i*$) **endif**
elseif get-nth (i, lst) $= \texttt{0}$ **then** $j*$
else strrchr* (add ($\texttt{32}$, $i*$, $\texttt{1}$), $1 + i$, n, lst, ch, $j*$) **endif**
else $j*$ **endif**

DEFINITION:
strpbrk* ($i1*$, $i1$, $n1$, $lst1$, $n2$, $lst2$)
$=$ **if** $i1 < n1$
then if get-nth ($i1$, $lst1$) $= \texttt{0}$ **then f**
elseif strchr1 ($\texttt{0}$, $n2$, $lst2$, get-nth ($i1$, $lst1$)) **then** $i1*$
else strpbrk* (add ($\texttt{32}$, $i1*$, $\texttt{1}$), $1 + i1$, $n1$, $lst1$, $n2$, $lst2$) **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 $j \in \mathbf{N}$
then **if** strncmp (*len*, mcdr (1 + *j*, *lst1*), mcdr (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, mcdr (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**

```
; strxfrm.
```

DEFINITION:
strxfrm1 (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 strxfrm1 ($1 + i$, put-nth (get-nth (i, *lst2*), i, *lst1*), *lst2*, $n - 1$) **endif**

DEFINITION:
strxfrm (*lst1*, *lst2*, n)
= **if** $n \simeq 0$ **then** *lst1*
 else strxfrm1 (0, *lst1*, *lst2*, n) **endif**

```
; a list of characters.
```

DEFINITION:
lst-of-chrp (*lst*)
= **if** listp (*lst*)
 then (car (*lst*) $\in$ **N**)
 $\land$ (car (*lst*) < 256)
 $\land$ lst-of-chrp (cdr (*lst*))
 else t endif

```
; theorems about lst-of-chrp.
```

THEOREM: get-lst-of-chrp
lst-of-chrp (*lst*) $\rightarrow$ ((get-nth (i, *lst*) < 256) $\land$ (get-nth (i, *lst*) $\in$ **N**))

THEOREM: put-lst-of-chrp
lst-of-chrp (*lst*)
$\rightarrow$ (lst-of-chrp (put-nth (x, i, *lst*)) = (($x \in$ **N**) $\land$ (x < 256)))

```
#|
(prove-lemma lessp-read-mem-1 (rewrite)
     (lessp (read-mem x mem 1) 256)
     ((use (byte-read-nat-rangep (n 8)))
```

```
      (enable nat-rangep)))

(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 $(i,\ n,\ lst) = (\text{slen}\,(i,\ n,\ lst) < n)$

```
; events for slen, which is part of stringp.
```

THEOREM: slen-ubound
$(i \leq n) \rightarrow (n \not< \text{slen}\,(i,\ n,\ lst))$

THEOREM: slen-lbound
$\text{slen}\,(i,\ n,\ lst) \not< i$

THEOREM: slen-01
$(\text{slen}\,(i,\ \mathtt{0},\ lst) = \text{fix}\,(i))$
$\wedge$ (slen $(i,\ \mathtt{1},\ lst)$
  $=$ **if** $(i < \mathtt{1}) \wedge (\text{get-nth}\,(\mathtt{0},\ lst) \neq \mathtt{0})$ **then** $\mathtt{1}$
    else fix (i) **endif**)

THEOREM: slen-add1
slen $(i,\ 1 + i,\ lst)$
$=$ **if** get-nth $(i,\ lst) = \mathtt{0}$ **then** fix (i)
  else $1 + i$ **endif**

THEOREM: slen-put0
$\text{slen}\,(i,\ n,\ \text{put-nth}\,(\mathtt{0},\ i,\ lst)) = \text{fix}\,(i)$

THEOREM: slen-put
$(j < i) \rightarrow (\text{slen}\,(i,\ n,\ \text{put-nth}\,(v,\ j,\ lst)) = \text{slen}\,(i,\ n,\ lst))$

THEOREM: lessp-slen-mcdr
$(\text{slen}\,(i + k,\ n1,\ lst1) < n1)$
$\rightarrow\quad ((\text{slen}\,(i,\ n,\ \text{mcdr}\,(k,\ lst1)) < (n1\ -\ k)) = \mathbf{t})$

THEOREM: lessp-slen-mcdr-0
$(\text{slen}\,(1 + i,\ n1,\ lst1) < n1)$
$\rightarrow\quad ((\text{slen}\,(i,\ n,\ \text{mcdr}\,(\mathtt{1},\ lst1)) < (n1\ - 1)) = \mathbf{t})$

THEOREM: slen-rec
$((\text{get-nth}\,(i,\ lst) \neq \mathtt{0}) \land (i < n))$
$\rightarrow\quad (\text{slen}\,(1 + i,\ n,\ lst) = \text{slen}\,(i,\ n,\ lst))$

EVENT: Disable slen.

```
; theorems about stringp.
```

THEOREM: stringp-la
$(\text{stringp}\,(i,\ n,\ lst) \land \text{int-rangep}\,(n,\ nn) \land (\text{get-nth}\,(i,\ lst) \neq \mathtt{0}))$
$\rightarrow\quad \text{int-rangep}\,(1 + i,\ nn)$

```
; theorems about strchr.
```

THEOREM: strchr-bounds
$(n \not< \text{strchr}\,(i,\ n,\ lst,\ ch))$
$\land\quad (\text{strchr}\,(i,\ n,\ lst,\ ch) \rightarrow (\text{strchr}\,(i,\ n,\ lst,\ ch) \not< i))$

THEOREM: memchr-bounds
$((i + n) \not< \text{memchr1}\,(i,\ n,\ lst,\ ch))$
$\land\quad (\text{memchr1}\,(i,\ n,\ lst,\ ch) \rightarrow (\text{memchr1}\,(i,\ n,\ lst,\ ch) \not< i))$

THEOREM: strpbrk-bounds
$(n1 \not< \text{strpbrk}\,(i1,\ n1,\ lst1,\ n2,\ lst2))$
$\land\quad (\text{strpbrk}\,(i1,\ n1,\ lst1,\ n2,\ lst2) \rightarrow (\text{strpbrk}\,(i1,\ n1,\ lst1,\ n2,\ lst2) \not< i1))$

THEOREM: strrchr-bounds
$(j \leq n) \rightarrow (n \not< \text{strrchr}\,(i,\ n,\ lst,\ ch,\ j))$

THEOREM: strstr-bounds
$(n1 \not< \text{strstr1}\,(i,\ n1,\ lst1,\ n2,\ lst2,\ len))$
$\land\quad (\text{strstr1}\,(i,\ n1,\ lst1,\ n2,\ lst2,\ len)$
$\qquad\rightarrow\quad (\text{strstr1}\,(i,\ n1,\ lst1,\ n2,\ lst2,\ len) \not< i))$

THEOREM: strspn-ubound
$(n1 \not\simeq \mathtt{0}) \rightarrow (\text{strspn}\,(i,\ n1,\ lst1,\ n2,\ lst2) < n1)$

THEOREM: strspn-bounds
$(n1 \not< \text{strspn}(i, n1, lst1, n2, lst2))$
$\wedge \quad (\text{strspn}(i, n1, lst1, n2, lst2) \rightarrow (\text{strspn}(i, n1, lst1, n2, lst2) \not< i))$

THEOREM: strcspn-bounds
$(n1 \not< \text{strcspn}(i, n1, lst1, n2, lst2))$
$\wedge \quad (\text{strcspn}(i, n1, lst1, n2, lst2) \rightarrow (\text{strcspn}(i, n1, lst1, n2, lst2) \not< i))$

```
; a lemma to establish nat-rangep.
```

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

EVENT: Disable nat-rangep-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
$(\text{disjoint}(a, m, b, n)$
$\wedge \quad (j \leq m)$
$\wedge \quad ((\text{nat-to-int}(k, 32) + l) \leq n)$
$\wedge \quad (\text{nat-to-int}(k, 32) \in \mathbf{N}))$
$\rightarrow \quad \text{disjoint}(a, j, \text{add}(32, b, k), l)$

THEOREM: disjoint-2-int
$(\text{disjoint}(a, m, b, n)$
$\wedge \quad ((\text{nat-to-int}(i, 32) + j) \leq m)$
$\wedge \quad (l \leq n)$
$\wedge \quad (\text{nat-to-int}(i, 32) \in \mathbf{N}))$
$\rightarrow \quad \text{disjoint}(\text{add}(32, a, i), j, b, l)$

THEOREM: disjoint-2~int
$(\text{disjoint}(a, m, b, n)$
$\wedge \quad ((\text{nat-to-int}(i, 32) + j) \leq m)$
$\wedge \quad (l \leq n)$
$\wedge \quad (\text{nat-to-int}(i, 32) \in \mathbf{N}))$
$\rightarrow \quad \text{disjoint}(b, l, \text{add}(32, a, i), j)$

THEOREM: disjoint-3-int
$(\text{disjoint}(a, m, b, n)$
$\wedge \quad ((\text{nat-to-int}(i, 32) + j) \leq m)$
$\wedge \quad ((\text{nat-to-int}(k, 32) + l) \leq n)$
$\wedge \quad (\text{nat-to-int}(i, 32) \in \mathbf{N})$
$\wedge \quad (\text{nat-to-int}(k, 32) \in \mathbf{N}))$
$\rightarrow \quad \text{disjoint}(\text{add}(32, a, i), j, \text{add}(32, b, k), l)$

THEOREM: read-memp-ram1-uint
(ram-addrp (*addr*, *mem*, *k*) $\wedge$ ((nat-to-uint (*i*) + *j*) $\leq$ *k*))
$\rightarrow$ read-memp (add (32, *addr*, *i*), *mem*, *j*)

THEOREM: write-memp-ram1-uint
(ram-addrp (*addr*, *mem*, *k*) $\wedge$ ((nat-to-uint (*i*) + *j*) $\leq$ *k*))
$\rightarrow$ write-memp (add (32, *addr*, *i*), *mem*, *j*)

THEOREM: disjoint-1-uint
(disjoint (*a*, *m*, *b*, *n*) $\wedge$ ($j \leq m$) $\wedge$ ((nat-to-uint (*k*) + *l*) $\leq$ *n*))
$\rightarrow$ disjoint (*a*, *j*, add (32, *b*, *k*), *l*)

THEOREM: disjoint-2-uint
(disjoint (*a*, *m*, *b*, *n*) $\wedge$ ((nat-to-uint (*i*) + *j*) $\leq$ *m*) $\wedge$ ($l \leq n$))
$\rightarrow$ disjoint (add (32, *a*, *i*), *j*, *b*, *l*)

THEOREM: disjoint-2~-uint
(disjoint (*a*, *m*, *b*, *n*) $\wedge$ ((nat-to-uint (*i*) + *j*) $\leq$ *m*) $\wedge$ ($l \leq n$))
$\rightarrow$ disjoint (*b*, *l*, add (32, *a*, *i*), *j*)

THEOREM: disjoint-3-uint
(disjoint (*a*, *m*, *b*, *n*)
$\wedge$ ((nat-to-uint (*i*) + *j*) $\leq$ *m*)
$\wedge$ ((nat-to-uint (*k*) + *l*) $\leq$ *n*))
$\rightarrow$ disjoint (add (32, *a*, *i*), *j*, add (32, *b*, *k*), *l*)

THEOREM: disjoint-5-uint
(disjoint (add (32, *a*, *i*), *m*, *b*, *n*)
$\wedge$ ((*j* + index-n (0, *i*)) $\leq$ *m*)
$\wedge$ ((nat-to-uint (*k*) + *l*) $\leq$ *n*))
$\rightarrow$ disjoint (*a*, *j*, add (32, *b*, *k*), *l*)

THEOREM: disjoint-6-uint
(disjoint (add (32, *a*, *i*), *m*, *b*, *n*)
$\wedge$ ((*j* + index-n (*i1*, *i*)) $\leq$ *m*)
$\wedge$ ($l \leq n$))
$\rightarrow$ disjoint (add (32, *a*, *i1*), *j*, *b*, *l*)

THEOREM: disjoint-7-uint
(disjoint (add (32, *a*, *i*), *m*, *b*, *n*)
$\wedge$ ((*j* + index-n (*i1*, *i*)) $\leq$ *m*)
$\wedge$ ((nat-to-uint (*k*) + *l*) $\leq$ *n*))
$\rightarrow$ disjoint (add (32, *a*, *i1*), *j*, add (32, *b*, *k*), *l*)

THEOREM: read-rn-int-8_32
nat-to-int (read-rn (8, *rn*, *rfile*), 32) = nat-to-uint (read-rn (8, *rn*, *rfile*))

THEOREM: read-rn-int-16_32
nat-to-int (read-rn (16, *rn*, *rfile*), 32) = nat-to-uint (read-rn (16, *rn*, *rfile*))

THEOREM: read-rn-int-8_16
nat-to-int (read-rn (8, *rn*, *rfile*), 16) = nat-to-uint (read-rn (8, *rn*, *rfile*))

THEOREM: read-mem-int-8_32
nat-to-int (read-mem (x, *mem*, 1), 32) = nat-to-uint (read-mem (x, *mem*, 1))

THEOREM: read-mem-int-16_32
nat-to-int (read-mem (x, *mem*, 2), 32) = nat-to-uint (read-mem (x, *mem*, 2))

THEOREM: read-mem-int-8_16
nat-to-int (read-mem (x, *mem*, 1), 16) = nat-to-uint (read-mem (x, *mem*, 1))

THEOREM: idifference-int-rangep
$((x \in \mathbf{N}) \wedge (y \in \mathbf{N}) \wedge$ int-rangep $(x,\ n) \wedge$ int-rangep $(y,\ n))$
$\rightarrow$ int-rangep (idifference $(x,\ y),\ n)$

EVENT: Disable idifference.

```
; some more arithmetic.
```

THEOREM: difference-cancel-1
$(i \leq j)$
$\rightarrow \quad ((((i + j) - (2 * i)) = (j - i))$
$\quad\quad \wedge \quad (((2 * j) - (i + j)) = (j - i)))$

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
$(i \leq j) \rightarrow (((j - i) \div 2) \not< ((j - 1) - ((i + j) \div 2)))$

THEOREM: plus-0
$(0 + x) = \mathrm{fix}\,(x)$

THEOREM: plus-times-sub1
$(x + (x * (y - 1)))$
$=$ **if** $y \simeq 0$ **then** fix (x)
else $x * y$ **endif**

EVENT: Disable plus-times-sub1.

THEOREM: plus2-times-sub1
$(x + y + (x * (z - 1)))$
$= \ \ \textbf{if } z \simeq 0 \textbf{ then } x + y$
$\quad\ \textbf{else } y + (x * z) \textbf{ endif}$

THEOREM: plus3-times-sub1
$(x + y + z + (x * (z1 - 1)))$
$= \ \ \textbf{if } z1 \simeq 0 \textbf{ then } x + y + z$
$\quad\ \textbf{else } y + z + (x * z1) \textbf{ endif}$

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
$((a - c) < (c * (b - 1)))$
$= \ \ \textbf{if } c \le a \textbf{ then } a < (c * b)$
$\quad\ \textbf{else } 1 < b \textbf{ endif}$

THEOREM: difference-lessp-cancel-2
$(c \le a)$
$\rightarrow \ (((a - c) < (d + (c * (b - 1))))$
$\quad = \ \ \textbf{if } b \simeq 0 \textbf{ then } (a - c) < d$
$\quad\quad\ \textbf{else } a < (d + (c * b)) \textbf{ endif})$

```
; two funny lemmas.  But seems more useful than add-uint!
```

THEOREM: add-uintxx
$(\text{nat-rangep}(x, n)$
$\wedge \ \ \text{nat-rangep}(y, n)$
$\wedge \ \ ((\text{nat-to-uint}(x) + \text{nat-to-uint}(y)) < \exp(2, n)))$
$\rightarrow \ (\text{nat-to-uint}(\text{add}(n, x, y)) = (\text{nat-to-uint}(x) + \text{nat-to-uint}(y)))$

THEOREM: add-uintxxx
$(\text{nat-rangep}(x, n)$
$\wedge \ \ \text{nat-rangep}(y, n)$
$\wedge \ \ ((\text{nat-to-uint}(x) + \text{nat-to-uint}(y)) \not< \exp(2, n)))$
$\rightarrow \ (\text{nat-to-uint}(\text{add}(n, x, y))$
$\quad = \ \ ((\text{nat-to-uint}(x) + \text{nat-to-uint}(y)) - \exp(2, n)))$

EVENT: Disable add-uintxxx.

THEOREM: ram-addrp-3
(ram-addrp (add ($\texttt{32}$, $addr$, i), mem, k) $\wedge$ ((index-n (h, i) $+$ j) $\leq$ k))
$\rightarrow$ ram-addrp (add ($\texttt{32}$, $addr$, h), mem, j)

THEOREM: ram-addrp-4
(ram-addrp ($addr$, mem, k) $\wedge$ ((nat-to-uint (h) $+$ j) $\leq$ k))
$\rightarrow$ ram-addrp (add ($\texttt{32}$, $addr$, h), mem, j)

THEOREM: and-z-commutativity
and-z (n, x, y) $=$ and-z (n, y, x)

```
; finally, establish the database.
```

EVENT: Make the library `"mc20-2"` and compile it.

Index