```
#|
 Copyright (C) 1985 by Warren A. Hunt, Jr.  All Rights Reserved.

 You may copy and distribute verbatim copies of this Nqthm-1992 event script as
 you receive it, in any medium, including embedding it verbatim in derivative
 works, provided that you conspicuously and appropriately publish on each copy
 a valid copyright notice "Copyright (C) 1985 by Warren A. Hunt, Jr.  All
 Rights Reserved."

 NO WARRANTY

 Warren A. Hunt, Jr. 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 Warren A. Hunt, Jr. 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.

|#

;;; -*- Mode: LISP; Base: 10; Syntax: Zetalisp -*-

;   The Mechanical Verification of the FM8501 Microprocessor

;                               by
;                         Warren A. Hunt, Jr.
```

EVENT: Start with the initial **thm** theory.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                              ;;
;;                               lemmas about PLUS                              ;;
;;                                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

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

THEOREM: plus-1
$(\mathtt{1} + x) = (1 + x)$

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

THEOREM: plus-add1
$(x + (1 + y))$
$= \quad$ **if** $y \in \mathbf{N}$ **then** $1 + (x + y)$
else $1 + x$ **endif**

THEOREM: commutativity2-of-plus
$(x + (y + z)) = (y + (x + z))$

THEOREM: commutativity-of-plus
$(x + y) = (y + x)$

THEOREM: associativity-of-plus
$((x + y) + z) = (x + (y + z))$

THEOREM: plus-equal-0
$((a + b) = \mathtt{0}) = ((a \simeq \mathtt{0}) \wedge (b \simeq \mathtt{0}))$

THEOREM: plus-cancellation
$((a + b) = (a + c)) = (\text{fix}\,(b) = \text{fix}\,(c))$

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                              ;;
;;                            lemmas about TIMES                                ;;
;;                                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: times-zero2
$(y \notin \mathbf{N}) \rightarrow ((x * y) = \mathtt{0})$

THEOREM: distributivity-of-times-over-plus
$(x * (y + z)) = ((x * y) + (x * z))$

THEOREM: times-add1
$(x * (1 + y))$
$= \quad$ **if** $y \in \mathbf{N}$ **then** $x + (x * y)$
else $\text{fix}\,(x)$ **endif**

THEOREM: commutativity-of-times
$(x * y) = (y * x)$

THEOREM: commutativity2-of-times
$(x * (y * z)) = (y * (x * z))$

THEOREM: associativity-of-times
$((x * y) * z) = (x * (y * z))$

THEOREM: equal-times-0
$((x * y) = \mathtt{0}) = ((x \simeq \mathtt{0}) \lor (y \simeq \mathtt{0}))$

THEOREM: times-1
$(\mathtt{1} * x) = \mathrm{fix}(x)$

THEOREM: equal-bools
$(((\mathit{bool1} = \mathbf{t}) \lor (\mathit{bool1} = \mathbf{f})) \land ((\mathit{bool2} = \mathbf{t}) \lor (\mathit{bool2} = \mathbf{f})))$
$\rightarrow \quad ((\mathit{bool1} = \mathit{bool2}) = ((\mathit{bool1} \rightarrow \mathit{bool2}) \land (\mathit{bool2} \rightarrow \mathit{bool1})))$

EVENT: Disable equal-bools.

THEOREM: lessp-times
$((y * x) < (x * z)) = ((x \not\simeq \mathtt{0}) \land (y < z))$

THEOREM: times-2-not-1
$(\mathtt{2} * x) \neq \mathtt{1}$

THEOREM: lessp-crock1
$((z + (\mathtt{2} * v * w)) < w) = ((v \simeq \mathtt{0}) \land (z < w))$

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                                    ;;
;;                                  lemmas about EXP                                  ;;
;;                                                                                    ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:
$\exp(i, j)$
$= \quad$ **if** $j \simeq \mathtt{0}$ **then** $\mathtt{1}$
else $i * \exp(i, j - 1)$ **endif**

THEOREM: exp-plus
$\exp(i, j + k) = (\exp(i, j) * \exp(i, k))$

THEOREM: exp-of-0
$\exp(0, k)$
$= \textbf{if } k \simeq 0 \textbf{ then } 1$
$\quad \textbf{else } 0 \textbf{ endif}$

THEOREM: exp-of-1
$\exp(1, k) = 1$

THEOREM: exp-by-0
$\exp(x, 0) = 1$

THEOREM: exp-times
$\exp(i * j, k) = (\exp(i, k) * \exp(j, k))$

THEOREM: exp-2-never-0
$0 < \exp(2, i)$

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                                 ;;
;;                            lemmas about DIFFERENCE                              ;;
;;                                                                                 ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: difference-elim
$((y \in \mathbf{N}) \wedge (y \not< x)) \rightarrow ((x + (y - x)) = y)$

THEOREM: difference-2
$(x - 2) = ((x - 1) - 1)$

THEOREM: difference-x-x
$(x - x) = 0$

THEOREM: difference-plus
$((j + x) - j) = \text{fix}(x)$

THEOREM: difference-plus-cancellation
$((a + x) - (a + y)) = (x - y)$

THEOREM: pathological-difference
$(x < y) \rightarrow ((x - y) = 0)$

THEOREM: difference-crock1
$((x + (y - z)) - y)$
$= \textbf{if } y < z \textbf{ then } x - y$
$\quad \textbf{else } x - z \textbf{ endif}$

THEOREM: difference-difference
$((x - y) - z) = (x - (y + z))$

THEOREM: lessp-difference
$((x - y) < x) = ((x \not\simeq 0) \wedge (y \not\simeq 0))$

THEOREM: difference-add1
$((1 + x) - y)$
$= \ \textbf{if } y < (1 + x) \textbf{ then } 1 + (x - y)$
$\quad \textbf{else } 0 \textbf{ endif}$

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                              ;;
;;                     lemmas about QUOTIENT and REMAINDER                      ;;
;;                                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: remainder-quotient
$((x \textbf{ mod } y) + (y * (x \div y))) = \mathrm{fix}(x)$

THEOREM: remainder-by-1
$(y \textbf{ mod } 1) = 0$

THEOREM: remainder-by-nonnumber
$(x \notin \mathbf{N}) \rightarrow ((y \textbf{ mod } x) = \mathrm{fix}(y))$

THEOREM: lessp-remainder
$((x \textbf{ mod } y) < y) = (y \not\simeq 0)$

THEOREM: remainder-quotient-elim
$((y \not\simeq 0) \wedge (x \in \mathbf{N})) \rightarrow (((x \textbf{ mod } y) + (y * (x \div y))) = x)$

THEOREM: remainder-x-x
$(x \textbf{ mod } x) = 0$

THEOREM: remainder-plus
$((j + x) \textbf{ mod } j) = (x \textbf{ mod } j)$

THEOREM: remainder-plus-times
$((x + (i * j)) \textbf{ mod } j) = (x \textbf{ mod } j)$

THEOREM: remainder-plus-times-commuted
$((x + (j * i)) \textbf{ mod } j) = (x \textbf{ mod } j)$

THEOREM: remainder-times
$((j * i) \textbf{ mod } j) = 0$

THEOREM: remainder-add1-times
$((1 + (j * i))\ \mathbf{mod}\ j) = (\mathtt{1}\ \mathbf{mod}\ j)$

THEOREM: quotient-plus-times
$((x + (i * j)) \div j)$
$=$ **if** $j \simeq \mathtt{0}$ **then** $\mathtt{0}$
else $i + (x \div j)$ **endif**

THEOREM: quotient-plus-times-commuted
$((x + (j * i)) \div j)$
$=$ **if** $j \simeq \mathtt{0}$ **then** $\mathtt{0}$
else $i + (x \div j)$ **endif**

THEOREM: quotient-times
$((j * i) \div j)$
$=$ **if** $j \simeq \mathtt{0}$ **then** $\mathtt{0}$
else $\mathrm{fix}\,(i)$ **endif**

THEOREM: quotient-add1-times
$((1 + (j * i)) \div j)$
$=$ **if** $j \simeq \mathtt{0}$ **then** $\mathtt{0}$
else $i + (\mathtt{1} \div j)$ **endif**

EVENT: Disable times.

THEOREM: times-distributes-over-remainder
$((x * y)\ \mathbf{mod}\ (x * z)) = (x * (y\ \mathbf{mod}\ z))$

THEOREM: remainder-of-1
$(\mathtt{1}\ \mathbf{mod}\ x)$
$=$ **if** $x = \mathtt{1}$ **then** $\mathtt{0}$
else $\mathtt{1}$ **endif**

THEOREM: remainder-crock1
$((x \in \mathbf{N}) \wedge (x < z) \wedge (v \in \mathbf{N}) \wedge (z \in \mathbf{N}) \wedge (z \neq \mathtt{0}))$
$\rightarrow$ $(((1 + ((\mathtt{2} * x) + (\mathtt{2} * v * z)))\ \mathbf{mod}\ (\mathtt{2} * z))$
$=$ $(1 + (\mathtt{2} * x)))$

THEOREM: times-2-distributes-over-remainder-add1
$((1 + (\mathtt{2} * y))\ \mathbf{mod}\ (\mathtt{2} * z)) = (1 + (\mathtt{2} * (y\ \mathbf{mod}\ z)))$

THEOREM: remainder-crock2
$(v < \exp(\mathtt{2},\ size - 1))$
$\rightarrow$ $((((\mathtt{2} * v) + (\mathtt{2} * w * \exp(\mathtt{2},\ size - 1)))$
$\mathbf{mod}$ $(\mathtt{2} * \exp(\mathtt{2},\ size - 1)))$
$=$ $(\mathtt{2} * v))$

THEOREM: remainder-crock3

$((x + (y - z)) \textbf{ mod } y)$
$= \textbf{ if } (x + (y - z)) < y \textbf{ then } x + (y - z)$
$\quad \textbf{elseif } y < z \textbf{ then } x \textbf{ mod } y$
$\quad \textbf{else } (x - z) \textbf{ mod } y \textbf{ endif}$

THEOREM: remainder-of-0

$(0 \textbf{ mod } x) = 0$

THEOREM: remainder-crock4

$((1 + (x + (y - z))) \textbf{ mod } y)$
$= \textbf{ if } (1 + (x + (y - z))) < y \textbf{ then } 1 + (x + (y - z))$
$\quad \textbf{elseif } y < z \textbf{ then } (1 + x) \textbf{ mod } y$
$\quad \textbf{elseif } z < (1 + x) \textbf{ then } (1 + (x - z)) \textbf{ mod } y$
$\quad \textbf{else } 0 \textbf{ endif}$

THEOREM: quotient-2i-by-2

$(((1 + (i + i)) \div 2) = \text{fix}(i)) \land (((i + i) \div 2) = \text{fix}(i))$

THEOREM: remainder-2i-by-2

$(((1 + (i + i)) \textbf{ mod } 2) = 1) \land (((i + i) \textbf{ mod } 2) = 0)$

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;;                            boolean operations                              ;;
;;                                                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:

$\text{xor}(x, y)$
$= \textbf{ if } x$
$\quad \textbf{then if } y \textbf{ then f}$
$\qquad \textbf{else t endif}$
$\quad \textbf{elseif } y \textbf{ then t}$
$\quad \textbf{else f endif}$

DEFINITION:

$\text{b-not}(x)$
$= \textbf{ if } x \textbf{ then f}$
$\quad \textbf{else t endif}$

DEFINITION:

$\text{b-nand}(x, y)$
$= \textbf{ if } x$
$\quad \textbf{then if } y \textbf{ then f}$
$\qquad \textbf{else t endif}$
$\quad \textbf{else t endif}$

DEFINITION:
b-nor (x, y)
= **if** x **then f**
elseif y **then f**
else t endif

DEFINITION: b-and $(x, y) = (x \wedge y)$

DEFINITION: b-or $(x, y) = (x \vee y)$

DEFINITION:
b-xor (x, y)
= **if** x
then if y **then f**
else t endif
elseif y **then t**
else f endif

DEFINITION:
b-equv (x, y)
= **if** x
then if y **then t**
else f endif
elseif y **then f**
else t endif

DEFINITION: boolp (x) = (truep (x) $\vee$ falsep (x))

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;;                      primitive bit vector operations                       ;;
;;                                                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

EVENT: Add the shell *bitv*, with bottom object function symbol *btm*, with recognizer function symbol *bitvp*, and 2 accessors: *bit*, with type restriction (one-of truep falsep) and default value false; *vec*, with type restriction (one-of bitvp) and default value btm.

THEOREM: boolp-bit
boolp (bit (x))

DEFINITION:
fix-bv (x)
= **if** bitvp (x) **then** x
else BTM **endif**

DEFINITION:
carry (c)
= **if** c **then** `1`
else `0` **endif**

DEFINITION:
size (a)
= **if** bitvp (a)
then if $a =$ BTM **then** `0`
else $1 +$ size (vec (a)) **endif**
else `0` **endif**

DEFINITION:
trunc $(a,\ n)$
= **if** $n \simeq$ `0` **then** BTM
else bitv (bit (a), trunc (vec (a), $n - 1$)) **endif**

DEFINITION:
compl (x)
= **if** bitvp (x)
then if $x =$ BTM **then** BTM
else bitv $(\neg$ bit (x), compl (vec (x))) **endif**
else BTM **endif**

```
; IT MAY BE POSSIBLE TO ELIMINATE THE C FLG.
```

DEFINITION:
incr $(c,\ x)$
= **if** bitvp (x)
then if $x =$ BTM **then** BTM
else bitv (xor $(c,$ bit $(x))$, incr $(c \wedge$ bit $(x),$ vec $(x)))$ **endif**
else BTM **endif**

DEFINITION:
bitn $(x,\ n)$
= **if** $n \simeq$ `0` **then f**
elseif $n =$ `1` **then** bit (x)
else bitn (vec (x), $n - 1$) **endif**

DEFINITION:
btmp (a)
= **if** bitvp (a) **then** $a =$ BTM
else t endif

DEFINITION:
all-zerosp (a)
= **if** bitvp (a)
then if $a =$ BTM **then t**
else $(\neg$ bit $(a)) \wedge$ all-zerosp (vec (a)) **endif**
else t endif

DEFINITION:
all-onesp (a)
= **if** bitvp (a)
then if $a =$ BTM **then t**
else bit $(a) \wedge$ all-onesp (vec (a)) **endif**
else t endif

THEOREM: size-of-trunc
size (trunc (a, n)) = fix (n)

THEOREM: size-of-compl
size (compl (a)) = size (a)

THEOREM: size-of-incr
size (incr (c, a)) = size (a)

THEOREM: incr-f-noop
incr (**f**, a) = fix-bv (a)

THEOREM: size-0
(size $(x) = 0$) = btmp (x)

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                              ;;
;;            Definitions about logical functions                               ;;
;;                                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:
v-not (a)
= **if** bitvp (a)
then if $a =$ BTM **then** BTM
else bitv $(\neg$ bit (a), v-not (vec (a))) **endif**
else BTM **endif**

DEFINITION:
v-or $(a,\ b)$
= **if** bitvp (a)
then if $a =$ BTM **then** BTM
else bitv (bit $(a) \vee$ bit (b), v-or (vec (a), vec (b))) **endif**
else BTM **endif**

DEFINITION:
v-and $(a,\ b)$
= **if** bitvp (a)
then if $a =$ BTM **then** BTM
else bitv (bit $(a) \wedge$ bit (b), v-and (vec (a), vec (b))) **endif**
else BTM **endif**

DEFINITION:
v-xor $(a,\ b)$
= **if** bitvp (a)
then if $a =$ BTM **then** BTM
else bitv (xor (bit (a), bit (b)), v-xor (vec (a), vec (b))) **endif**
else BTM **endif**

DEFINITION:
bv-not (a)
= **if** bitvp (a)
then if $a =$ BTM **then** BTM
else bitv (b-not (bit (a)), bv-not (vec (a))) **endif**
else BTM **endif**

DEFINITION:
bv-or $(a,\ b)$
= **if** bitvp (a)
then if $a =$ BTM **then** BTM
else bitv (b-or (bit (a), bit (b)), bv-or (vec (a), vec (b))) **endif**
else BTM **endif**

DEFINITION:
bv-and $(a,\ b)$
= **if** bitvp (a)
then if $a =$ BTM **then** BTM
else bitv (b-and (bit (a), bit (b)), bv-and (vec (a), vec (b))) **endif**
else BTM **endif**

DEFINITION:
bv-xor $(a,\ b)$
= **if** bitvp (a)

then if $a = \text{BTM}$ **then** BTM
else bitv (b-xor (bit (a), bit (b)), bv-xor (vec (a), vec (b))) **endif**
else BTM **endif**

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                              ;;
;;                   TC numbers and their operations                            ;;
;;                                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:
tcp (x) = (($x \in \mathbf{N}$) $\vee$ (negativep (x) $\wedge$ (negative-guts (x) $\not\simeq$ 0)))

DEFINITION:
tc-in-rangep (x, n)
= **if** $n \simeq 0$ **then f**
elseif negativep (x) **then** exp (2, $n - 1$) $\not<$ negative-guts (x)
else $x <$ exp (2, $n - 1$) **endif**

DEFINITION:
add (x, y)
= **if** negativep (x)
then if negativep (y) **then** $-$ (negative-guts (x) + negative-guts (y))
elseif $y <$ negative-guts (x) **then** $-$ (negative-guts (x) $- y$)
else $y -$ negative-guts (x) **endif**
elseif negativep (y)
then if $x <$ negative-guts (y) **then** $-$ (negative-guts (y) $- x$)
else $x -$ negative-guts (y) **endif**
else $x + y$ **endif**

THEOREM: commutativity2-of-add
add (x, add (y, z)) = add (y, add (x, z))

THEOREM: commutativity-of-add
add (x, y) = add (y, x)

THEOREM: associativity-of-add
add (add (x, y), z) = add (x, add (y, z))

THEOREM: add-with-carry-of-negatives-is-negative
(tcp (x) $\wedge$ tcp (y) $\wedge$ negativep (x) $\wedge$ negativep (y))
$\rightarrow$ negativep (add (x, add (y, carry (c))))

THEOREM: add-with-carry-of-non-negatives-is-non-negative

$$(\text{tcp}(x) \land \text{tcp}(y) \land (\neg\ \text{negativep}(x)) \land (\neg\ \text{negativep}(y)))$$
$$\rightarrow\ (\neg\ \text{negativep}(\text{add}(x,\ \text{add}(y,\ \text{carry}(c)))))$$

THEOREM: overflow-lemma1

$$\begin{aligned}&(\text{tcp}(x)\\ &\ \land\ \text{tcp}(y)\\ &\ \land\ \text{tc-in-rangep}(x,\ n)\\ &\ \land\ \text{tc-in-rangep}(y,\ n)\\ &\ \land\ \text{negativep}(\text{add}(x,\ \text{add}(y,\ \text{carry}(c)))))\\ &\rightarrow\ (\neg\ \text{negativep}(\text{add}(x,\ \text{add}(y,\ \text{add}(\text{carry}(c),\ \exp(2,\ n))))))\end{aligned}$$

THEOREM: overflow-lemma2

$$\begin{aligned}&(\text{tcp}(x)\\ &\ \land\ \text{tcp}(y)\\ &\ \land\ \text{tc-in-rangep}(x,\ n)\\ &\ \land\ \text{tc-in-rangep}(y,\ n)\\ &\ \land\ (\neg\ \text{negativep}(\text{add}(x,\ \text{add}(y,\ \text{carry}(c))))))\\ &\rightarrow\ \text{negativep}(\text{add}(x,\ \text{add}(y,\ \text{add}(\text{carry}(c),\ -\ \exp(2,\ n)))))\end{aligned}$$

THEOREM: opposite-signs-implies-tc-in-rangep

$$\begin{aligned}&(\text{tcp}(x)\\ &\ \land\ \text{tcp}(y)\\ &\ \land\ \text{tc-in-rangep}(x,\ n)\\ &\ \land\ \text{tc-in-rangep}(y,\ n)\\ &\ \land\ \text{negativep}(x)\\ &\ \land\ (\neg\ \text{negativep}(y)))\\ &\rightarrow\ \text{tc-in-rangep}(\text{add}(x,\ \text{add}(y,\ \text{carry}(c))),\ n)\end{aligned}$$

THEOREM: opposite-signs-implies-tc-in-rangep-commuted

$$\begin{aligned}&(\text{tcp}(x)\\ &\ \land\ \text{tcp}(y)\\ &\ \land\ \text{tc-in-rangep}(x,\ n)\\ &\ \land\ \text{tc-in-rangep}(y,\ n)\\ &\ \land\ \text{negativep}(x)\\ &\ \land\ (\neg\ \text{negativep}(y)))\\ &\rightarrow\ \text{tc-in-rangep}(\text{add}(y,\ \text{add}(x,\ \text{carry}(c))),\ n)\end{aligned}$$

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                              ;;
;;              Mapping Functions between BV, NAT, and TC                       ;;
;;                                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:
nat-to-bv (n, $size$)
= **if** $size \simeq$ 0 **then** BTM
else bitv (**if** (n **mod** 2) $\simeq$ 0 **then** **f**
else **t** **endif**,
nat-to-bv ($n \div 2$, $size - 1$)) **endif**

DEFINITION:
bv-to-nat (x)
= **if** bitvp (x)
then **if** x = BTM **then** 0
else **if** bit (x) **then** 1
else 0 **endif**
+ (2 $*$ bv-to-nat (vec (x))) **endif**
else 0 **endif**

DEFINITION:
tc-to-bv (x, $size$)
= **if** negativep (x) **then** incr (**t**, compl (nat-to-bv (negative-guts (x), $size$)))
else nat-to-bv (x, $size$) **endif**

DEFINITION:
bv-to-tc (x)
= **if** bitn (x, size (x)) **then** $-$ bv-to-nat (incr (**t**, compl (x)))
else bv-to-nat (x) **endif**

DEFINITION:
nat-to-tc (n, $size$)
= **if** $n <$ exp (2, $size - 1$) **then** n
else $-$ (exp (2, $size$) $- n$) **endif**

DEFINITION:
tc-to-nat (n, $size$)
= **if** negativep (n) **then** exp (2, $size$) $-$ negative-guts (n)
else n **endif**

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                          ;;
;;                  Elementary Properties of Mapping Functions              ;;
;;                                                                          ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

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

THEOREM: size-of-tc-to-bv
size (tc-to-bv (x, *size*)) = fix (*size*)

THEOREM: bv-to-nat-of-incr
bv-to-nat (incr (**t**, a))
= **if** all-onesp (a) **then** 0
else 1 + bv-to-nat (a) **endif**

THEOREM: bitn-implies-compl-not-all-onesp
bitn (a, i) → (¬ all-onesp (compl (a)))

THEOREM: upper-bound-on-bv-to-nat
bv-to-nat (a) < exp (2, size (a))

THEOREM: bv-to-nat-of-compl
bv-to-nat (compl (a)) = ((exp (2, size (a)) − 1) − bv-to-nat (a))

THEOREM: bv-to-nat-to-tc-lemma1
bitn (a, size (a))
→ (bv-to-nat (incr (**t**, compl (a))) = (exp (2, size (a)) − bv-to-nat (a)))

THEOREM: bv-to-nat-to-tc-lemma2
bitn (a, size (a))
= **if** size (a) ≃ 0 **then f**
else bv-to-nat (a) ≮ exp (2, size (a) − 1) **endif**

THEOREM: bv-to-nat-of-trunc
bv-to-nat (trunc (a, n)) = (bv-to-nat (a) **mod** exp (2, n))

THEOREM: all-onesp-of-compl
all-onesp (compl (a)) = (bv-to-nat (a) = 0)

THEOREM: tcp-bv-to-tc
tcp (bv-to-tc (a))

THEOREM: upper-bound-on-non-negative-bv-to-nat
(bitvp (a) ∧ (a ≠ BTM) ∧ (¬ bitn (a, size (a))))
→ (bv-to-nat (a) < exp (2, size (a) − 1))

THEOREM: lower-bound-on-negative-bv-to-nat
(bitvp (a) ∧ (a ≠ BTM) ∧ bitn (a, size (a)))
→ (bv-to-nat (a) ≮ exp (2, size (a) − 1))

THEOREM: tc-in-rangep-of-bv-to-tc
(n = size (a))
→ (tc-in-rangep (bv-to-tc (a), n) = (bitvp (a) ∧ (a ≠ BTM)))

THEOREM: bitn-means-negativep
$(n = \text{size}(a)) \rightarrow (\text{bitn}(a, n) = \text{negativep}(\text{bv-to-tc}(a)))$

EVENT: Disable bitn-means-negativep.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                                ;;
;;             Relationship between TC and NAT addition                           ;;
;;                                                                                ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: plus-to-add
$(\text{tcp}(x) \wedge \text{tcp}(y) \wedge \text{tc-in-rangep}(x, n) \wedge \text{tc-in-rangep}(y, n))$
$\rightarrow \quad (\text{nat-to-tc}((\text{carry}(c) + \text{tc-to-nat}(x, n) + \text{tc-to-nat}(y, n))$
mod $\text{exp}(2, n)$,
$n)$
$= \quad$ **if** $\text{tc-in-rangep}(\text{add}(x, \text{add}(y, \text{carry}(c))), n)$
then $\text{add}(x, \text{add}(y, \text{carry}(c)))$
elseif $\text{negativep}(\text{add}(x, \text{add}(y, \text{carry}(c))))$
then $\text{add}(x, \text{add}(y, \text{add}(\text{carry}(c), \text{exp}(2, n))))$
else $\text{add}(x, \text{add}(y, \text{add}(\text{carry}(c), -\text{exp}(2, n))))$ **endif**)

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                                ;;
;;       Fundamental Relationships between the Mapping Functions                  ;;
;;                                                                                ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: bv-to-nat-to-bv
$(\textit{size} = \text{size}(a)) \rightarrow (\text{nat-to-bv}(\text{bv-to-nat}(a), \textit{size}) = \text{fix-bv}(a))$

THEOREM: bv-to-nat-to-tc
$\text{bv-to-tc}(a) = \text{nat-to-tc}(\text{bv-to-nat}(a), \text{size}(a))$

EVENT: Disable bv-to-nat-to-tc.

THEOREM: nat-to-bv-to-nat
$\text{bv-to-nat}(\text{nat-to-bv}(n, \textit{size})) = (n \textbf{ mod } \text{exp}(2, \textit{size}))$

THEOREM: tc-to-bv-to-nat
$(\text{tcp}(x) \wedge \text{tc-in-rangep}(x, n))$
$\rightarrow \quad (\text{tc-to-nat}(x, n) = \text{bv-to-nat}(\text{tc-to-bv}(x, n)))$

EVENT: Disable tc-to-bv-to-nat.

THEOREM: bv-to-tc-to-nat
$(n = \text{size}(a)) \rightarrow (\text{tc-to-nat}(\text{bv-to-tc}(a),\ n) = \text{bv-to-nat}(a))$

EVENT: Disable bv-to-tc-to-nat.

THEOREM: incr-compl-nat-to-bv-0
$\text{incr}(\mathbf{t},\ \text{compl}(\text{nat-to-bv}(0,\ \text{size}(x)))) = \text{nat-to-bv}(0,\ \text{size}(x))$

THEOREM: bitn-on-implies-non-0
$\text{bitn}(a,\ n) \rightarrow (\text{bv-to-nat}(a) \neq 0)$

THEOREM: bv-to-tc-to-bv
$(\mathit{size} = \text{size}(a)) \rightarrow (\text{tc-to-bv}(\text{bv-to-tc}(a),\ \mathit{size}) = \text{fix-bv}(a))$

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                              ;;
;;                                Bridge Lemmas                                 ;;
;;                                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: embed-in-nat-to-bv
$(x = y) \rightarrow (\text{nat-to-bv}(x,\ n) = \text{nat-to-bv}(y,\ n))$

THEOREM: embed-in-tc-to-bv
$(x = y) \rightarrow (\text{tc-to-bv}(x,\ n) = \text{tc-to-bv}(y,\ n))$

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                              ;;
;;                                BV-ADDER                                      ;;
;;                                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:
bv-adder $(c,\ a,\ b)$
= **if** bitvp (a)
  then if a = BTM **then** bitv $(c,$ BTM$)$
    else bitv (b-xor $(c,$ b-xor (bit (a), bit (b))),
      bv-adder (b-or (b-and (bit (a), bit (b)),
        b-or (b-and (bit (a), c),

b-and (bit (*b*), *c*))),
vec (*a*),
vec (*b*))) **endif**
else bitv (*c*, BTM) **endif**

DEFINITION:
bv-adder-output (*c*, *a*, *b*) = trunc (bv-adder (*c*, *a*, *b*), size (*a*))

DEFINITION:
bv-adder-carry-out (*c*, *a*, *b*) = bitn (bv-adder (*c*, *a*, *b*), 1 + size (*a*))

DEFINITION:
bv-adder-overflowp (*c*, *a*, *b*)
= b-and (b-equv (bitn (*a*, size (*a*)), bitn (*b*, size (*b*))),
b-xor (bitn (*a*, size (*a*)), bitn (bv-adder-output (*c*, *a*, *b*), size (*a*))))

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                              ;;
;;                    Fundamental Properties of BV-ADDER                        ;;
;;                                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: size-of-bv-adder
size (bv-adder (*c*, *a*, *b*)) = (1 + size (*a*))

THEOREM: bv-to-nat-of-bv-adder
(bitvp (*a*) ∧ bitvp (*b*) ∧ (size (*a*) = size (*b*)) ∧ boolp (*c*))
→ (bv-to-nat (bv-adder (*c*, *a*, *b*))
= (bv-to-nat (*a*) + bv-to-nat (*b*) + carry (*c*)))

```
; From here on we use only the above two lemmas about BV-ADDER
; The definition of BV-ADDER is disabled and henceforth any adder
; with the two properties above would suffice.
```

EVENT: Disable bv-adder.

THEOREM: bv-adder-non-btm
bv-adder (*c*, *a*, *b*) ≠ BTM

THEOREM: size-bv-adder-output
size (bv-adder-output (*c*, *a*, *b*)) = size (*a*)

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;;                      NAT view of BV-ADDER                                  ;;
;;                                                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: nat-interpretation-of-bv-adder-output

$$
\begin{array}{l}
(\text{bitvp}(a) \wedge \text{bitvp}(b) \wedge (\text{size}(a) = \text{size}(b)) \wedge \text{boolp}(c)) \\
\rightarrow \quad (\text{bv-to-nat}(\text{bv-adder-output}(c,\ a,\ b)) \\
\qquad = \quad \textbf{if } (\text{bv-to-nat}(a) + \text{bv-to-nat}(b) + \text{carry}(c)) \\
\qquad\qquad\quad < \quad \exp(2, \text{size}(a)) \\
\qquad\qquad \textbf{then } \text{bv-to-nat}(a) + \text{bv-to-nat}(b) + \text{carry}(c) \\
\qquad\qquad \textbf{else } (\text{bv-to-nat}(a) + \text{bv-to-nat}(b) + \text{carry}(c)) \\
\qquad\qquad\qquad \textbf{mod} \quad \exp(2, \text{size}(a)) \textbf{ endif})
\end{array}
$$

EVENT: Disable nat-interpretation-of-bv-adder-output.

THEOREM: nat-bv-adder-output-seen-as-bit-vector

$$
\begin{array}{l}
(\text{bitvp}(a) \wedge \text{bitvp}(b) \wedge (\text{size}(a) = \text{size}(b)) \wedge \text{boolp}(c)) \\
\rightarrow \quad (\text{bv-adder-output}(c,\ a,\ b) \\
\qquad = \quad \textbf{if } (\text{bv-to-nat}(a) + \text{bv-to-nat}(b) + \text{carry}(c)) \\
\qquad\qquad\quad < \quad \exp(2, \text{size}(a)) \\
\qquad\qquad \textbf{then } \text{nat-to-bv}(\text{bv-to-nat}(a) + \text{bv-to-nat}(b) + \text{carry}(c), \\
\qquad\qquad\qquad\qquad\qquad \text{size}(a)) \\
\qquad\qquad \textbf{else } \text{nat-to-bv}((\text{bv-to-nat}(a) \\
\qquad\qquad\qquad\qquad\qquad + \quad \text{bv-to-nat}(b) \\
\qquad\qquad\qquad\qquad\qquad + \quad \text{carry}(c)) \\
\qquad\qquad\qquad\qquad\quad \textbf{mod} \quad \exp(2, \text{size}(a)), \\
\qquad\qquad\qquad\qquad\quad \text{size}(a)) \textbf{ endif})
\end{array}
$$

EVENT: Disable nat-bv-adder-output-seen-as-bit-vector.

THEOREM: nat-interpretation-of-bv-adder-carry-out

$$
\begin{array}{l}
(\text{bitvp}(a) \wedge \text{bitvp}(b) \wedge (\text{size}(a) = \text{size}(b)) \wedge \text{boolp}(c)) \\
\rightarrow \quad (\text{bv-adder-carry-out}(c,\ a,\ b) \\
\qquad = \quad ((\text{bv-to-nat}(a) + \text{bv-to-nat}(b) + \text{carry}(c)) \\
\qquad\qquad \not< \quad \exp(2, \text{size}(a))))
\end{array}
$$

EVENT: Disable nat-interpretation-of-bv-adder-carry-out.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;;                                                                                ;;
;;                          TC view of BV-ADDER                                   ;;
;;                                                                                ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: tc-interpretation-of-bv-adder-output-lemma1
(bitvp (*a*) ∧ bitvp (*b*) ∧ (size (*a*) = size (*b*)) ∧ boolp (*c*))
→ (bv-to-tc (trunc (bv-adder (*c*, *a*, *b*), size (*a*)))
= nat-to-tc ((carry (*c*) + bv-to-nat (*a*) + bv-to-nat (*b*))
mod exp (2, size (*a*)),
size (*a*)))

THEOREM: tc-interpretation-of-bv-adder-output
(bitvp (*a*)
∧ bitvp (*b*)
∧ (*a* ≠ BTM)
∧ (*b* ≠ BTM)
∧ (size (*a*) = size (*b*))
∧ boolp (*c*))
→ (bv-to-tc (bv-adder-output (*c*, *a*, *b*))
= **if** tc-in-rangep (add (bv-to-tc (*a*), add (bv-to-tc (*b*), carry (*c*))),
size (*a*))
then add (bv-to-tc (*a*), add (bv-to-tc (*b*), carry (*c*)))
elseif negativep (add (bv-to-tc (*a*), add (bv-to-tc (*b*), carry (*c*))))
then add (bv-to-tc (*a*),
add (bv-to-tc (*b*), add (carry (*c*), exp (2, size (*a*)))))
else add (bv-to-tc (*a*),
add (bv-to-tc (*b*),
add (carry (*c*), − exp (2, size (*a*))))) **endif**)

EVENT: Disable tc-interpretation-of-bv-adder-output-lemma1.

EVENT: Disable tc-interpretation-of-bv-adder-output.

THEOREM: tc-bv-adder-output-seen-as-bit-vector-lemma1
(bitvp (*a*)
∧ bitvp (*b*)
∧ (*a* ≠ BTM)
∧ (*b* ≠ BTM)
∧ (size (*a*) = size (*b*))
∧ boolp (*c*))
→ (tc-to-bv (bv-to-tc (bv-adder-output (*c*, *a*, *b*)),
size (bv-adder-output (*c*, *a*, *b*)))

```
    =   tc-to-bv (if tc-in-rangep (add (bv-to-tc (a),
                                        add (bv-to-tc (b), carry (c))),
                                   size (a))
                 then add (bv-to-tc (a), add (bv-to-tc (b), carry (c)))
                 elseif negativep (add (bv-to-tc (a),
                                        add (bv-to-tc (b), carry (c))))
                 then add (bv-to-tc (a),
                           add (bv-to-tc (b),
                                add (carry (c), exp (2, size (a)))))
                 else add (bv-to-tc (a),
                           add (bv-to-tc (b),
                                add (carry (c), - exp (2, size (a))))) endif,
                 size (a)))
```

EVENT: Disable tc-bv-adder-output-seen-as-bit-vector-lemma1.

THEOREM: tc-bv-adder-output-seen-as-bit-vector

```
(bitvp (a)
 ∧   bitvp (b)
 ∧   (a ≠ BTM)
 ∧   (b ≠ BTM)
 ∧   (size (a) = size (b))
 ∧   boolp (c))
→   (bv-adder-output (c, a, b)
     =   tc-to-bv (if tc-in-rangep (add (bv-to-tc (a),
                                         add (bv-to-tc (b), carry (c))),
                                    size (a))
                  then add (bv-to-tc (a), add (bv-to-tc (b), carry (c)))
                  elseif negativep (add (bv-to-tc (a),
                                         add (bv-to-tc (b), carry (c))))
                  then add (bv-to-tc (a),
                            add (bv-to-tc (b),
                                 add (carry (c), exp (2, size (a)))))
                  else add (bv-to-tc (a),
                            add (bv-to-tc (b),
                                 add (carry (c), - exp (2, size (a))))) endif,
                  size (a)))
```

EVENT: Disable tc-bv-adder-output-seen-as-bit-vector.

THEOREM: tc-interpretation-of-bv-adder-overflowp

```
(bitvp (a)
 ∧   bitvp (b)
```

$\wedge$ (size (*a*) = size (*b*))
$\wedge$ boolp (*c*)
$\wedge$ (*a* $\neq$ BTM)
$\wedge$ (*b* $\neq$ BTM))
$\rightarrow$ (bv-adder-overflowp (*c*, *a*, *b*)
= ($\neg$ tc-in-rangep (add (bv-to-tc (*a*), add (bv-to-tc (*b*), carry (*c*))),
size (*a*))))

EVENT: Disable tc-interpretation-of-bv-adder-overflowp.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                              ;;
;;            Definitions and lemmas about subtraction functions                ;;
;;                                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;(defn bv-subtracter (c a b)    ; B - A - C
;  (bv-adder (b-not c)
;            (bv-not a)
;            b))
```

DEFINITION:
bv-subtracter-output (*c*, *a*, *b*) = bv-adder-output (b-not (*c*), bv-not (*a*), *b*)

DEFINITION:
bv-subtracter-carry-out (*c*, *a*, *b*)
= b-not (bv-adder-carry-out (b-not (*c*), bv-not (*a*), *b*))

DEFINITION:
bv-subtracter-overflowp (*c*, *a*, *b*) = bv-adder-overflowp (b-not (*c*), bv-not (*a*), *b*)

THEOREM: bv-not-is-compl
bv-not (*a*) = compl (*a*)

THEOREM: subtract-lemma1
(size (*b*) = size (*a*))
$\rightarrow$ (((bv-to-nat (*b*) $-$ bv-to-nat (*a*)) $<$ exp (2, size (*a*))) = **t**)

THEOREM: subtract-lemma2
((*b* $\not\simeq$ 0) $\wedge$ ((*b* $-$ 1) $\not<$ *a*)) $\rightarrow$ (((*b* + ((*n* $-$ 1) $-$ *a*)) $<$ *n*) = **f**)

THEOREM: subtract-lemma3
((*b* $\not\simeq$ 0) $\wedge$ ((*b* $-$ 1) $\not<$ *a*) $\wedge$ (*a* $<$ *n*))
$\rightarrow$ (((*b* + ((*n* $-$ 1) $-$ *a*)) $-$ *n*) = ((*b* $-$ 1) $-$ *a*))

THEOREM: subtract-lemma4
(size (b) = size (a))
$\rightarrow$ ((((bv-to-nat $(b) - 1$) $-$ bv-to-nat (a)) $<$ exp (2, size (a))) = **t**)

THEOREM: size-bv-subtracter-output
size (bv-subtracter-output $(c,\ a,\ b)$) = size (a)

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;;                    NAT view of BV-SUBTRACTER                               ;;
;;                                                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: nat-interpretation-of-bv-subtracter-output
(bitvp (a) $\wedge$ bitvp (b) $\wedge$ (size (a) = size (b)) $\wedge$ boolp (c))
$\rightarrow$ (bv-to-nat (bv-subtracter-output $(c,\ a,\ b)$)
  = **if** bv-to-nat (b) $\not<$ (bv-to-nat (a) + carry (c))
   then bv-to-nat (b) $-$ (bv-to-nat (a) + carry (c))
   else exp (2, size (a))
    $-$ ((bv-to-nat (a) + carry (c))
     $-$ bv-to-nat (b)) **endif**)

EVENT: Disable nat-interpretation-of-bv-subtracter-output.

THEOREM: nat-bv-subtracter-output-seen-as-bit-vector
(bitvp (a) $\wedge$ bitvp (b) $\wedge$ (size (a) = size (b)) $\wedge$ boolp (c))
$\rightarrow$ (bv-subtracter-output $(c,\ a,\ b)$
  = **if** bv-to-nat (b) $\not<$ (bv-to-nat (a) + carry (c))
   then nat-to-bv (bv-to-nat (b) $-$ (bv-to-nat (a) + carry (c)),
    size (a))
   else nat-to-bv (exp (2, size (a))
    $-$ ((bv-to-nat (a) + carry (c))
     $-$ bv-to-nat (b)),
    size (a)) **endif**)

EVENT: Disable nat-bv-subtracter-output-seen-as-bit-vector.

THEOREM: equal-lessp-hack
$((x < y) = z)$
= **if** $x < y$ **then** $z =$ **t**
 else $z =$ **f** **endif**

THEOREM: nat-interpretation-of-bv-subtracter-carry-out
$(\text{bitvp}(a) \land \text{bitvp}(b) \land (\text{size}(a) = \text{size}(b)) \land \text{boolp}(c))$
$\rightarrow \quad (\text{bv-subtracter-carry-out}(c,\ a,\ b)$
$\quad = \quad (\text{bv-to-nat}(b) < (\text{bv-to-nat}(a) + \text{carry}(c))))$

EVENT: Disable nat-interpretation-of-bv-subtracter-carry-out.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;;                         TC view of BV-SUBTRACTER                           ;;
;;                                                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:
tc-minus (x)
$=$ **if** negativep (x) **then** negative-guts (x)
elseif $x \simeq 0$ **then** 0
else $-\ x$ **endif**

THEOREM: tc-minus-add
$\text{tc-minus}(\text{add}(x,\ y)) = \text{add}(\text{tc-minus}(x),\ \text{tc-minus}(y))$

THEOREM: bitn-compl
$((n \not\simeq 0) \land (\text{size}(a) \not< n)) \rightarrow (\text{bitn}(\text{compl}(a),\ n) = (\neg\ \text{bitn}(a,\ n)))$

```
;(prove-lemma bitn-on-implies-non-0 (rewrite)
;  (implies (bitn a n) (not (equal (bv-to-nat a) 0))))
```

THEOREM: equal-difference-0
$((x - y) = 0) = (y \not< x)$

THEOREM: top-bit-off-implies-smaller
$(\text{bitvp}(a) \land (a \neq \text{BTM}) \land (\neg\ \text{bitn}(a,\ \text{size}(a))))$
$\rightarrow \quad (\text{bv-to-nat}(a) < (\exp(2,\ \text{size}(a)) - 1))$

THEOREM: tc-minus-bv-to-tc
$(\text{bitvp}(a) \land (a \neq \text{BTM}))$
$\rightarrow \quad (\text{tc-minus}(\text{bv-to-tc}(a))$
$\quad = \quad$ **if** $\text{bv-to-tc}(a) = 0$ **then** 0
else $\text{add}(1,\ \text{bv-to-tc}(\text{compl}(a)))$ **endif**)

DEFINITION:
tc-fix (x)
= **if** tcp (x) **then** x
else `0` **endif**

THEOREM: tcp-add
(tcp (x) → tcp (add (x, y))) ∧ (tcp (y) → tcp (add (x, y)))

THEOREM: add-0
add (`0`, x) = tc-fix (x)

THEOREM: add-1-1
add (`1`, add (`-1`, x)) = tc-fix (x)

THEOREM: bv-to-tc-compl-0
(bitvp (a) ∧ (a ≠ BTM) ∧ (bv-to-tc (a) = `0`))
→ (bv-to-tc (compl (a)) = `-1`)

THEOREM: tc-minus-add-carry
(bitvp (a) ∧ (a ≠ BTM))
→ (tc-minus (add (bv-to-tc (a), carry (c)))
= add (bv-to-tc (compl (a)), carry (¬ c)))

THEOREM: boolp-b-not
boolp (b-not (x))

THEOREM: compl-not-btm
(compl (a) = BTM) = ((¬ bitvp (a)) ∨ (a = BTM))

THEOREM: tc-interpretation-of-bv-subtracter-output
(bitvp (a)
∧ bitvp (b)
∧ (a ≠ BTM)
∧ (b ≠ BTM)
∧ (size (a) = size (b))
∧ boolp (c))
→ (bv-to-tc (bv-subtracter-output (c, a, b))
= **if** tc-in-rangep (add (bv-to-tc (b),
tc-minus (add (bv-to-tc (a), carry (c)))),
size (a))
then add (bv-to-tc (b), tc-minus (add (bv-to-tc (a), carry (c))))
elseif negativep (add (bv-to-tc (b),
tc-minus (add (bv-to-tc (a), carry (c)))))
then add (bv-to-tc (b),
add (exp (`2`, size (a)),

tc-minus (add (bv-to-tc (a), carry (c)))))
else add (bv-to-tc (b),
add (− exp (2, size (a)),
tc-minus (add (bv-to-tc (a), carry (c))))) **endif**)

EVENT: Disable tc-interpretation-of-bv-subtracter-output.

THEOREM: tc-bv-subtracter-output-seen-as-bit-vector-lemma1
(bitvp (a)
∧ bitvp (b)
∧ ($a \neq$ BTM)
∧ ($b \neq$ BTM)
∧ (size (a) = size (b))
∧ boolp (c))
→ (tc-to-bv (bv-to-tc (bv-subtracter-output (c, a, b)),
size (bv-subtracter-output (c, a, b)))
= tc-to-bv (**if** tc-in-rangep (add (bv-to-tc (b),
tc-minus (add (bv-to-tc (a),
carry (c)))),
size (a))
then add (bv-to-tc (b),
tc-minus (add (bv-to-tc (a), carry (c))))
elseif negativep (add (bv-to-tc (b),
tc-minus (add (bv-to-tc (a),
carry (c)))))
then add (bv-to-tc (b),
add (exp (2, size (a)),
tc-minus (add (bv-to-tc (a), carry (c)))))
else add (bv-to-tc (b),
add (− exp (2, size (a)),
tc-minus (add (bv-to-tc (a), carry (c))))) **endif**,
size (a)))

EVENT: Disable tc-bv-subtracter-output-seen-as-bit-vector-lemma1.

THEOREM: tc-bv-subtracter-output-seen-as-bit-vector
(bitvp (a)
∧ bitvp (b)
∧ ($a \neq$ BTM)
∧ ($b \neq$ BTM)
∧ (size (a) = size (b))
∧ boolp (c))
→ (bv-subtracter-output (c, a, b)

```
=   if tc-in-rangep (add (bv-to-tc (b),
                          tc-minus (add (bv-to-tc (a), carry (c)))),
                     size (a))
    then tc-to-bv (add (bv-to-tc (b),
                        tc-minus (add (bv-to-tc (a), carry (c)))),
                   size (a))
    elseif negativep (add (bv-to-tc (b),
                           tc-minus (add (bv-to-tc (a), carry (c)))))
    then tc-to-bv (add (bv-to-tc (b),
                        add (exp (2, size (a)),
                             tc-minus (add (bv-to-tc (a), carry (c))))),
                   size (a))
    else tc-to-bv (add (bv-to-tc (b),
                        add (- exp (2, size (a)),
                             tc-minus (add (bv-to-tc (a), carry (c))))),
                   size (a)) endif)
```

EVENT: Disable tc-bv-subtracter-output-seen-as-bit-vector.

THEOREM: tc-interpretation-of-bv-subtracter-overflowp

```
(bitvp (a)
 ∧  bitvp (b)
 ∧  (a ≠ BTM)
 ∧  (b ≠ BTM)
 ∧  (size (a) = size (b))
 ∧  boolp (c))
→  (bv-subtracter-overflowp (c, a, b)
    =  (¬ tc-in-rangep (add (bv-to-tc (b),
                             tc-minus (add (bv-to-tc (a), carry (c)))),
                        size (a))))
```

EVENT: Disable tc-interpretation-of-bv-subtracter-overflowp.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                              ;;
;;            Definitions and lemmas for shift and rotate functions             ;;
;;                                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:

```
v-append (a, b)
=   if bitvp (a)
```

then if a = BTM **then** b
else bitv (bit (a), v-append (vec (a), b)) **endif**
else b **endif**

DEFINITION:
v-lsr (a)
= **if** bitvp (a)
then if a = BTM **then** BTM
else v-append (vec (a), bitv (**f**, BTM)) **endif**
else BTM **endif**

DEFINITION:
v-lsl (a)
= **if** bitvp (a)
then if a = BTM **then** BTM
else v-append (bitv (**f**, BTM), trunc (a, size (a) − 1)) **endif**
else BTM **endif**

DEFINITION:
v-asr (a)
= **if** bitvp (a)
then if a = BTM **then** BTM
else v-append (vec (a), bitv (bitn (a, size (a)), BTM)) **endif**
else BTM **endif**

DEFINITION: v-asl (a) = v-lsl (a)

DEFINITION:
v-ror (a, c)
= **if** bitvp (a)
then if a = BTM **then** BTM
else v-append (vec (a), bitv (c, BTM)) **endif**
else BTM **endif**

DEFINITION:
v-rol (a, c)
= **if** bitvp (a)
then if a = BTM **then** BTM
else v-append (bitv (c, BTM), trunc (a, size (a) − 1)) **endif**
else BTM **endif**

DEFINITION: bv-append (a, b) = v-append (a, b)

DEFINITION: bv-lsr (a) = v-lsr (a)

DEFINITION: bv-asr (a) = v-asr (a)

DEFINITION: bv-ror $(a,\ c)$ = v-ror $(a,\ c)$

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                                    ;;
;;            Natural and two's complement interpretation of shifters                 ;;
;;                                                                                    ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: bv-to-nat-v-append
bv-to-nat (v-append $(a,\ b)$)
= (bv-to-nat (a) + (bv-to-nat (b) $*$ exp (2, size (a))))

THEOREM: bv-lsr-divides-by-two-lemma
v-lsr (a) = nat-to-bv (bv-to-nat (a) $\div$ 2, size (a))

DEFINITION:
mod2 (x)
= **if** negativep (x) **then** $-$ ((1 + negative-guts (x)) $\div$ 2)
else $x \div 2$ **endif**

THEOREM: bitvp-v-append
bitvp (v-append $(a,\ b)$)
= **if** bitvp (a) **then** $(a \neq \text{BTM}) \vee$ bitvp (b)
else bitvp (b) **endif**

THEOREM: bitn-v-append
bitn (v-append $(a,\ b)$, n)
= **if** size $(a) < n$ **then** bitn $(b,\ n -$ size $(a))$
else bitn $(a,\ n)$ **endif**

THEOREM: size-v-append
size (v-append $(a,\ b)$) = (size (a) + size (b))

THEOREM: difference-x-x-1
$(x - (x - 1))$
= **if** $x \simeq 0$ **then** 0
else 1 **endif**

THEOREM: bv-to-nat-of-twos-compl
bv-to-nat (incr (**t**, compl (a)))
= **if** bv-to-nat $(a) = 0$ **then** 0
else exp (2, size (a)) $-$ bv-to-nat (a) **endif**

THEOREM: difference-2x-x
$((2 * y) - (x + y)) = (y - x)$

THEOREM: quotient-crock1
$((1 + ((2 * y) - (2 * x))) \div 2) = (y - x)$

THEOREM: add1-difference-sub1
$(1 + ((x - 1) - y))$
$=$ **if** $y < x$ **then** $x - y$
else 1 **endif**

THEOREM: quotient-crock2
$(((2 * y) - (2 * x)) \div 2) = (y - x)$

THEOREM: bv-asr-divides-by-two
bv-to-tc (bv-asr (a)) = mod2 (bv-to-tc (a))

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;;             Definitions and lemmas about hardware if                       ;;
;;             and constructor for ALU result                                 ;;
;;                                                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:
b-if (c, a, b) = b-nand (b-nand (c, a), b-nand (b-not (c), b))

DEFINITION:
bv-if (c, a, b)
$=$ **if** bitvp (a)
then if a = BTM **then** BTM
else bitv (b-if (c, bit (a), bit (b)), bv-if (c, vec (a), vec (b))) **endif**
else BTM **endif**

THEOREM: bv-if-works
bv-if (c, a, b)
$=$ **if** c
then if bitvp (a) **then** a
else BTM **endif**
else trunc (b, size (a)) **endif**

EVENT: Add the shell *bv-cv*, with recognizer function symbol *bv-cvp* and 3 accessors: *bv*, with type restriction (one-of bitvp) and default value btm; *c*, with type restriction (one-of truep falsep) and default value false; *v*, with type restriction (one-of truep falsep) and default value false.

Definition:
bv-cv-if $(c,\ a,\ b)$
$=\quad$ bv-cv (bv-if $(c,$ bv $(a),$ bv $(b)),$ b-if $(c,$ c $(a),$ c $(b)),$ b-if $(c,$ v $(a),$ v $(b)))$

Theorem: trunc-size
$(\text{bitvp}\,(b) \land (\text{size}\,(b) = \text{size}\,(a))) \rightarrow (\text{trunc}\,(b,\ \text{size}\,(a)) = b)$

Theorem: bv-cv-if-works
$(\text{bv-cvp}\,(a) \land \text{bv-cvp}\,(b) \land (\text{size}\,(\text{bv}\,(a)) = \text{size}\,(\text{bv}\,(b))))$
$\rightarrow\quad ((\text{bv-cv-if}\,(\mathbf{t},\ a,\ b) = a) \land (\text{bv-cv-if}\,(\mathbf{f},\ a,\ b) = b))$

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                              ;;
;;            SIZE lemmas                                                       ;;
;;                                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

Theorem: size-bv-bv-cv-if
$\text{size}\,(\text{bv}\,(\text{bv-cv-if}\,(c,\ a,\ b))) = \text{size}\,(\text{bv}\,(a))$

```
; We rewrite "bv-not" to "v-not", etc., and so we need these size lemmas about
; the v-functions.
```

Theorem: size-v-not
$\text{size}\,(\text{v-not}\,(a)) = \text{size}\,(a)$

Theorem: size-v-and
$\text{size}\,(\text{v-and}\,(a,\ b)) = \text{size}\,(a)$

Theorem: size-v-or
$\text{size}\,(\text{v-or}\,(a,\ b)) = \text{size}\,(a)$

Theorem: size-v-xor
$\text{size}\,(\text{v-xor}\,(a,\ b)) = \text{size}\,(a)$

Theorem: size-v-append-1
size (**if** bitvp (a)
    then if $a =$ BTM **then** BTM
        else v-append (vec (a), bitv $(x,$ BTM$)$) **endif**
    else BTM **endif**)
$=\quad$ size (a)

THEOREM: size-v-append-2
size (v-append (vec (a), bitv (x, BTM)))
= **if** bitvp (a)
  then if a = BTM **then** 1
    else size (a) **endif**
  else 1 **endif**

```
;  We have already proved (SIZE (BV-ADDER-OUTPUT C A B)) = (SIZE A) and
;                         (SIZE (BV-SUBTRACTER-OUTPUT C A B)) = (SIZE A).
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                                  ;;
;;             ALU definition                                                       ;;
;;                                                                                  ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

EVENT: Disable bv-cv-if.

DEFINITION:
bv-alu-cv (*a*, *b*, *c*, *op-code*)
= bv-cv-if (bitn (*op-code*, 4),
        bv-cv-if (bitn (*op-code*, 3),
            bv-cv-if (bitn (*op-code*, 2),
                bv-cv-if (bitn (*op-code*, 1),
                    bv-cv (*a*, **f**, **f**),
                    bv-cv (bv-not (*a*), **f**, **f**)),
                bv-cv-if (bitn (*op-code*, 1),
                    bv-cv (bv-and (*a*, *b*), **f**, **f**),
                    bv-cv (bv-or (*a*, *b*), **f**, **f**))),
            bv-cv-if (bitn (*op-code*, 2),
                bv-cv-if (bitn (*op-code*, 1),
                    bv-cv (bv-xor (*a*, *b*), **f**, **f**),
                    bv-cv (bv-lsr (*a*), bitn (*a*, 1), **f**)),
                bv-cv-if (bitn (*op-code*, 1),
                    bv-cv (bv-asr (*a*), bitn (*a*, 1), **f**),
                    bv-cv (bv-ror (*a*, *c*),
                        if size (*a*) $\simeq$ 0 **then** *c*
                        else bitn (*a*, 1) **endif**,
                        f)))),
        bv-cv-if (bitn (*op-code*, 3),
            bv-cv-if (bitn (*op-code*, 2),
                bv-cv-if (bitn (*op-code*, 1),
                    bv-cv (bv-subtracter-output (**f**, *a*, *b*),
                        bv-subtracter-carry-out (**f**,
                                                 a,
                                                 b),
                        bv-subtracter-overflowp (**f**,
                                                 a,
                                                 b)),
                    bv-cv (bv-subtracter-output (*c*, *a*, *b*),
                        bv-subtracter-carry-out (*c*,

```
                                                    a,
                                                    b),
                                bv-subtracter-overflowp (c,
                                                         a,
                                                         b))),
                        bv-cv-if (bitn (op-code, 1),
                                  bv-cv (bv-subtracter-output (t,
                                                               nat-to-bv (0,
                                                                          size (a)),
                                                               a),
                                         bv-subtracter-carry-out (t,
                                                                  nat-to-bv (0,
                                                                             size (a)),
                                                                  a),
                                         bv-subtracter-overflowp (t,
                                                                  nat-to-bv (0,
                                                                             size (a)),
                                                                  a)),
                                  bv-cv (bv-subtracter-output (f,
                                                               a,
                                                               nat-to-bv (0,
                                                                          size (a))),
                                         bv-subtracter-carry-out (f,
                                                                  a,
                                                                  nat-to-bv (0,
                                                                             size (a))),
                                         bv-subtracter-overflowp (f,
                                                                  a,
                                                                  nat-to-bv (0,
                                                                             size (a))))))),
                bv-cv-if (bitn (op-code, 2),
                          bv-cv-if (bitn (op-code, 1),
                                    bv-cv (bv-adder-output (f, a, b),
                                           bv-adder-carry-out (f, a, b),
                                           bv-adder-overflowp (f, a, b)),
                                    bv-cv (bv-adder-output (c, a, b),
                                           bv-adder-carry-out (c, a, b),
                                           bv-adder-overflowp (c, a, b))),
                          bv-cv-if (bitn (op-code, 1),
                                    bv-cv (bv-adder-output (t,
                                                            a,
                                                            nat-to-bv (0,
                                                                       size (a))),
                                           bv-adder-carry-out (t,
```

```
                                          a,
                                          nat-to-bv (0,
                                                     size (a))),
                        bv-adder-overflowp (t,
                                          a,
                                          nat-to-bv (0,
                                                     size (a)))),
                     bv-cv (a, f, f)))))
```

THEOREM: v-not-is-compl
v-not (a) = compl (a)

THEOREM: bv-and-is-v-and
bv-and $(a,\ b)$ = v-and $(a,\ b)$

THEOREM: bv-or-is-v-or
bv-or $(a,\ b)$ = v-or $(a,\ b)$

THEOREM: bv-xor-is-v-xor
bv-xor $(a,\ b)$ = v-xor $(a,\ b)$

THEOREM: bv-cv-if-reduce
bv-cv-if $(c$, bv-cv $(x,\ y,\ z)$, bv-cv $(x,\ y,\ z))$ = bv-cv $(x,\ y,\ z)$

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;;          BV-ALU with theorem-prover "ifs".  This was done to speed the     ;;
;;          rewriting when proving the large lemmas that follow.              ;;
;;                                                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: bv-alu-cv-with-ifs

```
(bitvp (a) ∧ bitvp (b) ∧ (size (a) = size (b)))
→   (bv-alu-cv (a, b, c, op-code)
      =   if bitn (op-code, 4)
          then if bitn (op-code, 3)
               then if bitn (op-code, 2)
                    then if bitn (op-code, 1)  then bv-cv (a, f, f)
                         else bv-cv (bv-not (a), f, f) endif
                    elseif bitn (op-code, 1)  then bv-cv (bv-and (a, b), f, f)
                    else bv-cv (bv-or (a, b), f, f) endif
               elseif bitn (op-code, 2)
               then if bitn (op-code, 1)  then bv-cv (bv-xor (a, b), f, f)
```

```
            else bv-cv (bv-lsr (a), bitn (a, 1), f) endif
        elseif bitn (op-code, 1)
        then bv-cv (bv-asr (a), bitn (a, 1), f)
        else bv-cv (bv-ror (a, c),
                    if size (a) ≃ 0 then c
                    else bitn (a, 1) endif,
                    f) endif
elseif bitn (op-code, 3)
then if bitn (op-code, 2)
        then if bitn (op-code, 1)
                then bv-cv (bv-subtracter-output (f, a, b),
                            bv-subtracter-carry-out (f, a, b),
                            bv-subtracter-overflowp (f, a, b))
                else bv-cv (bv-subtracter-output (c, a, b),
                            bv-subtracter-carry-out (c, a, b),
                            bv-subtracter-overflowp (c, a, b)) endif
        elseif bitn (op-code, 1)
        then bv-cv (bv-subtracter-output (t,
                                          nat-to-bv (0, size (a)),
                                          a),
                    bv-subtracter-carry-out (t,
                                             nat-to-bv (0, size (a)),
                                             a),
                    bv-subtracter-overflowp (t,
                                             nat-to-bv (0, size (a)),
                                             a))
        else bv-cv (bv-subtracter-output (f,
                                          a,
                                          nat-to-bv (0,
                                                     size (a))),
                    bv-subtracter-carry-out (f,
                                             a,
                                             nat-to-bv (0,
                                                        size (a))),
                    bv-subtracter-overflowp (f,
                                             a,
                                             nat-to-bv (0,
                                                        size (a)))) endif
elseif bitn (op-code, 2)
then if bitn (op-code, 1)
        then bv-cv (bv-adder-output (f, a, b),
                    bv-adder-carry-out (f, a, b),
                    bv-adder-overflowp (f, a, b))
        else bv-cv (bv-adder-output (c, a, b),
```

bv-adder-carry-out $(c,\ a,\ b)$,
bv-adder-overflowp $(c,\ a,\ b)$) **endif**
elseif bitn ($op\text{-}code$, 1)
then bv-cv (bv-adder-output (**t**, a, nat-to-bv (0, size (a))),
bv-adder-carry-out (**t**, a, nat-to-bv (0, size (a))),
bv-adder-overflowp (**t**, a, nat-to-bv (0, size (a))))
else bv-cv (a, **f**, **f**) **endif**)

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                              ;;
;;         Boolean ALU help lemmas                                              ;;
;;                                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: nat-to-bv-of-2i
boolp (c)
$\rightarrow$ (nat-to-bv (carry (c) $+$ ($i + i$), n)
$=$ **if** $n \simeq 0$ **then** BTM
else bitv (c, nat-to-bv (i, $n - 1$)) **endif**)

THEOREM: nat-to-bv-to-nat-wrong-size
nat-to-bv (bv-to-nat (a), n) $=$ trunc (a, n)

THEOREM: nat-to-bv-ignores-remainder
nat-to-bv (i **mod** exp (2, n), n) $=$ nat-to-bv (i, n)

EVENT: Disable nat-to-bv-ignores-remainder.

THEOREM: bv-adder-as-shifter
(bitvp (a) $\wedge$ boolp (c))
$\rightarrow$ (bv-adder (c, a, a)
$=$ nat-to-bv (bv-to-nat (a) $+$ bv-to-nat (a) $+$ carry (c),
1 $+$ size (a)))

THEOREM: bv-adder-output-used-as-asl
(bitvp (a) $\wedge$ boolp (c))
$\rightarrow$ (bv-adder-output (c, a, a)
$=$ **if** $a =$ BTM **then** BTM
else bitv (c, trunc (a, size (a) $- 1$)) **endif**)

EVENT: Disable bv-adder-output-used-as-asl.

THEOREM: bitn-equiv-lessp
$\text{bitvp}(a) \rightarrow (\text{bitn}(a, \text{size}(a)) = (\text{bv-to-nat}(a) \not< \exp(2, \text{size}(a) - 1)))$

THEOREM: bv-adder-carry-out-used-as-asl
$(\text{bitvp}(a) \land \text{boolp}(c))$
$\rightarrow \quad (\text{bv-adder-carry-out}(c, a, a)$
$\quad = \quad \textbf{if } \text{size}(a) \simeq 0 \textbf{ then } c$
$\quad\quad \textbf{else } \text{bitn}(a, \text{size}(a)) \textbf{ endif})$

EVENT: Disable bv-adder-carry-out-used-as-asl.

EVENT: Disable bv-adder-as-shifter.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                             ;;
;;           Boolean operation of ALU                                          ;;
;;                                                                             ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: bv-alu-cv-correct-boolean
$(\text{bitvp}(a) \land \text{bitvp}(b) \land (\text{size}(a) = \text{size}(b)) \land \text{boolp}(c))$
$\rightarrow \quad ((\text{bv-alu-cv}(a, b, c, \text{nat-to-bv}(15, 4)) = \text{bv-cv}(a, \mathbf{f}, \mathbf{f}))$
$\quad \land \quad (\text{bv-alu-cv}(a, b, c, \text{nat-to-bv}(14, 4)) = \text{bv-cv}(\text{v-not}(a), \mathbf{f}, \mathbf{f}))$
$\quad \land \quad (\text{bv-alu-cv}(a, b, c, \text{nat-to-bv}(13, 4)) = \text{bv-cv}(\text{v-and}(a, b), \mathbf{f}, \mathbf{f}))$
$\quad \land \quad (\text{bv-alu-cv}(a, b, c, \text{nat-to-bv}(12, 4)) = \text{bv-cv}(\text{v-or}(a, b), \mathbf{f}, \mathbf{f}))$
$\quad \land \quad (\text{bv-alu-cv}(a, b, c, \text{nat-to-bv}(11, 4)) = \text{bv-cv}(\text{v-xor}(a, b), \mathbf{f}, \mathbf{f}))$
$\quad \land \quad (\text{bv-alu-cv}(a, b, c, \text{nat-to-bv}(10, 4))$
$\quad\quad = \quad \text{bv-cv}(\text{v-lsr}(a), \text{bitn}(a, 1), \mathbf{f}))$
$\quad \land \quad (\text{bv-alu-cv}(a, b, c, \text{nat-to-bv}(9, 4))$
$\quad\quad = \quad \text{bv-cv}(\text{v-asr}(a), \text{bitn}(a, 1), \mathbf{f}))$
$\quad \land \quad (\text{bv-alu-cv}(a, b, c, \text{nat-to-bv}(8, 4))$
$\quad\quad = \quad \text{bv-cv}(\text{v-ror}(a, c),$
$\quad\quad\quad \textbf{if } \text{size}(a) \simeq 0 \textbf{ then } c$
$\quad\quad\quad \textbf{else } \text{bitn}(a, 1) \textbf{ endif},$
$\quad\quad\quad \mathbf{f}))$
$\quad \land \quad ((a = b)$
$\quad\quad \rightarrow \quad ((\text{bv}(\text{bv-alu-cv}(a, b, c, \text{nat-to-bv}(3, 4))) = \text{v-lsl}(a))$
$\quad\quad\quad \land \quad (\text{c}(\text{bv-alu-cv}(a, b, c, \text{nat-to-bv}(3, 4)))$
$\quad\quad\quad\quad = \quad \text{bitn}(a, \text{size}(a)))))$
$\quad \land \quad ((a = b)$
$\quad\quad \rightarrow \quad ((\text{bv}(\text{bv-alu-cv}(a, b, c, \text{nat-to-bv}(3, 4))) = \text{v-asl}(a))$
$\quad\quad\quad \land \quad (\text{c}(\text{bv-alu-cv}(a, b, c, \text{nat-to-bv}(3, 4)))$

$$
\begin{array}{l}
\qquad\qquad\quad = \quad \text{bitn}\,(a,\ \text{size}\,(a)))))\\
\quad \wedge \quad ((a = b)\\
\qquad\quad \rightarrow \quad ((\text{bv}\,(\text{bv-alu-cv}\,(a,\ b,\ c,\ \text{nat-to-bv}\,(2,\ 4))) = \text{v-rol}\,(a,\ c))\\
\qquad\qquad\quad \wedge \quad (\text{c}\,(\text{bv-alu-cv}\,(a,\ b,\ c,\ \text{nat-to-bv}\,(2,\ 4)))\\
\qquad\qquad\qquad = \quad \textbf{if}\ \text{size}\,(a) \simeq 0\ \textbf{then}\ c\\
\qquad\qquad\qquad\quad\ \ \textbf{else}\ \text{bitn}\,(a,\ \text{size}\,(a))\ \textbf{endif}))))
\end{array}
$$

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                              ;;
;;           Natural number ALU help lemmas                                     ;;
;;                                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: v-append-quotient
$(\text{bitvp}\,(a) \wedge (a \neq \text{BTM}))$
$\rightarrow \quad (\text{nat-to-bv}\,(\text{bv-to-nat}\,(a) \div 2,\ \text{size}\,(a))$
$\quad = \quad \text{v-append}\,(\text{vec}\,(a),\ \text{bitv}\,(\mathbf{f},\ \text{BTM})))$

THEOREM: bit-interpretation-of-even-and-odd
$((\text{bv-to-nat}\,(a)\ \textbf{mod}\ 2) = 0) = (\neg\ \text{bit}\,(a))$

THEOREM: nat-to-bv-to-nat-hidden-to-accomodate-hands-off
$(x - \text{bv-to-nat}\,(\text{nat-to-bv}\,(0,\ n))) = \text{fix}\,(x)$

THEOREM: zero-not-less-than-anything
$(\text{bv-to-nat}\,(a) - 1) \not< \text{bv-to-nat}\,(\text{nat-to-bv}\,(0,\ \text{size}\,(a)))$

THEOREM: nat-interpretation-of-inc-bit-vector
$\text{bitvp}\,(a)$
$\rightarrow \quad (\text{bv-adder-output}\,(\mathbf{t},\ a,\ \text{nat-to-bv}\,(0,\ \text{size}\,(a)))$
$\quad = \quad \textbf{if}\ (1 + \text{bv-to-nat}\,(a)) < \exp\,(2,\ \text{size}\,(a))$
$\qquad\ \ \textbf{then}\ \text{nat-to-bv}\,(1 + \text{bv-to-nat}\,(a),\ \text{size}\,(a))$
$\qquad\ \ \textbf{else}\ \text{nat-to-bv}\,(0,\ \text{size}\,(a))\ \textbf{endif})$

EVENT: Disable nat-interpretation-of-inc-bit-vector.

THEOREM: nat-interpretation-of-inc-carry-out
$\text{bitvp}\,(a)$
$\rightarrow \quad (\text{bv-adder-carry-out}\,(\mathbf{t},\ a,\ \text{nat-to-bv}\,(0,\ \text{size}\,(a)))$
$\quad = \quad ((1 + \text{bv-to-nat}\,(a)) \not< \exp\,(2,\ \text{size}\,(a))))$

EVENT: Disable nat-interpretation-of-inc-carry-out.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                           ;;
;;           Natural number operation of ALU                                 ;;
;;                                                                           ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: bv-alu-cv-correct-natural-number

```
 (bitvp (a) ∧ bitvp (b) ∧ (size (a) = size (b)) ∧ boolp (c))
→   ((bv-alu-cv (a, b, c, nat-to-bv (10, 4))
        =   bv-cv (nat-to-bv (bv-to-nat (a) ÷ 2, size (a)),
                   (bv-to-nat (a) mod 2) ≄ 0,
                   f))
     ∧   (bv (bv-alu-cv (a, b, c, nat-to-bv (7, 4)))
           =   if bv-to-nat (b) ≮ bv-to-nat (a)
               then nat-to-bv (bv-to-nat (b) − bv-to-nat (a), size (a))
               else nat-to-bv (exp (2, size (a))
                               −   (bv-to-nat (a)
                                     −   bv-to-nat (b)),
                               size (a)) endif)
     ∧   (c (bv-alu-cv (a, b, c, nat-to-bv (7, 4)))
           =   (bv-to-nat (b) < bv-to-nat (a)))
     ∧   (bv (bv-alu-cv (a, b, c, nat-to-bv (6, 4)))
           =   if bv-to-nat (b) ≮ (bv-to-nat (a) + carry (c))
               then nat-to-bv (bv-to-nat (b)
                                −   (bv-to-nat (a) + carry (c)),
                               size (a))
               else nat-to-bv (exp (2, size (a))
                               −   ((bv-to-nat (a)
                                      +   carry (c))
                                     −   bv-to-nat (b)),
                               size (a)) endif)
     ∧   (c (bv-alu-cv (a, b, c, nat-to-bv (6, 4)))
           =   (bv-to-nat (b) < (bv-to-nat (a) + carry (c))))
     ∧   (bv (bv-alu-cv (a, b, c, nat-to-bv (5, 4)))
           =   if bv-to-nat (a) ≃ 0
               then nat-to-bv (exp (2, size (a)) − 1, size (a))
               else nat-to-bv (bv-to-nat (a) − 1, size (a)) endif)
     ∧   (c (bv-alu-cv (a, b, c, nat-to-bv (5, 4))) = (bv-to-nat (a) ≃ 0))
     ∧   (bv (bv-alu-cv (a, b, c, nat-to-bv (3, 4)))
           =   if (bv-to-nat (a) + bv-to-nat (b))
                   <   exp (2, size (a))
               then nat-to-bv (bv-to-nat (a) + bv-to-nat (b), size (a))
```

$$
\begin{aligned}
&\quad\quad\quad \textbf{else}\ \text{nat-to-bv}\,((\text{bv-to-nat}\,(a) + \text{bv-to-nat}\,(b)) \\
&\quad\quad\quad\quad\quad\quad \textbf{mod}\quad \exp\,(\mathtt{2}, \text{size}\,(a)), \\
&\quad\quad\quad\quad\quad\quad \text{size}\,(a))\ \textbf{endif}) \\
\land\quad & (\text{c}\,(\text{bv-alu-cv}\,(a,\ b,\ c,\ \text{nat-to-bv}\,(\mathtt{3},\ \mathtt{4}))) \\
&\ =\ ((\text{bv-to-nat}\,(a) + \text{bv-to-nat}\,(b)) \not< \exp\,(\mathtt{2}, \text{size}\,(a)))) \\
\land\quad & (\text{bv}\,(\text{bv-alu-cv}\,(a,\ b,\ c,\ \text{nat-to-bv}\,(\mathtt{2},\ \mathtt{4}))) \\
&\ =\ \textbf{if}\ (\text{bv-to-nat}\,(a) + \text{bv-to-nat}\,(b) + \text{carry}\,(c)) \\
&\quad\quad\quad < \quad \exp\,(\mathtt{2}, \text{size}\,(a)) \\
&\quad\quad\quad \textbf{then}\ \text{nat-to-bv}\,(\text{bv-to-nat}\,(a) \\
&\quad\quad\quad\quad\quad\quad + \quad \text{bv-to-nat}\,(b) \\
&\quad\quad\quad\quad\quad\quad + \quad \text{carry}\,(c), \\
&\quad\quad\quad\quad\quad\quad \text{size}\,(a)) \\
&\quad\quad\quad \textbf{else}\ \text{nat-to-bv}\,((\text{bv-to-nat}\,(a) \\
&\quad\quad\quad\quad\quad\quad + \quad \text{bv-to-nat}\,(b) \\
&\quad\quad\quad\quad\quad\quad + \quad \text{carry}\,(c)) \\
&\quad\quad\quad\quad\quad\quad \textbf{mod}\quad \exp\,(\mathtt{2}, \text{size}\,(a)), \\
&\quad\quad\quad\quad\quad\quad \text{size}\,(a))\ \textbf{endif}) \\
\land\quad & (\text{c}\,(\text{bv-alu-cv}\,(a,\ b,\ c,\ \text{nat-to-bv}\,(\mathtt{2},\ \mathtt{4}))) \\
&\ =\ ((\text{bv-to-nat}\,(a) + \text{bv-to-nat}\,(b) + \text{carry}\,(c)) \\
&\quad\quad\quad \not< \quad \exp\,(\mathtt{2}, \text{size}\,(a)))) \\
\land\quad & (\text{bv}\,(\text{bv-alu-cv}\,(a,\ b,\ c,\ \text{nat-to-bv}\,(\mathtt{1},\ \mathtt{4}))) \\
&\ =\ \textbf{if}\ (1 + \text{bv-to-nat}\,(a)) < \exp\,(\mathtt{2}, \text{size}\,(a)) \\
&\quad\quad\quad \textbf{then}\ \text{nat-to-bv}\,(1 + \text{bv-to-nat}\,(a), \text{size}\,(a)) \\
&\quad\quad\quad \textbf{else}\ \text{nat-to-bv}\,(\mathtt{0}, \text{size}\,(a))\ \textbf{endif}) \\
\land\quad & (\text{c}\,(\text{bv-alu-cv}\,(a,\ b,\ c,\ \text{nat-to-bv}\,(\mathtt{1},\ \mathtt{4}))) \\
&\ =\ ((1 + \text{bv-to-nat}\,(a)) \not< \exp\,(\mathtt{2}, \text{size}\,(a)))))
\end{aligned}
$$

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;;           Two's complement ALU help lemmas                                 ;;
;;                                                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: twos-complement-twos-complement
$\text{bitvp}\,(a) \rightarrow (\text{incr}\,(\mathbf{t}, \text{compl}\,(\text{incr}\,(\mathbf{t}, \text{compl}\,(a)))) = a)$

THEOREM: tc-to-bv-to-tc-negative
$(\text{bitvp}\,(a) \land (\text{size}\,(a) = n)) \rightarrow (\text{tc-to-bv}\,(\text{bv-to-tc}\,(a),\ n) = a)$

THEOREM: v-asr-is-a-bitv
$\text{bitvp}\,(\text{v-asr}\,(a))$

THEOREM: size-of-v-asr
$\text{size}\,(\text{v-asr}\,(a)) = \text{size}\,(a)$

THEOREM: bv-asr-divides-by-two-bridge
bitvp (*a*) → (tc-to-bv (mod2 (bv-to-tc (*a*)), size (*a*)) = bv-asr (*a*))

THEOREM: bv-to-tc-to-bv-of-zero-is-zero
bv-to-tc (tc-to-bv (0, *n*)) = 0

THEOREM: tc-to-bv-is-not-btm-for-non-zero-size
(*n* ≠ 0) → ((tc-to-bv (0, *n*) = BTM) = **f**)

THEOREM: nat-of-zero-is-tc-of-zero
nat-to-bv (0, *size*) = tc-to-bv (0, *size*)

THEOREM: tc-fix-of-add
tc-fix (add (*a*, *b*)) = add (tc-fix (*a*), *b*)

THEOREM: tc-to-bv-of-bitv-not-btm
((*a* ≠ BTM) ∧ bitvp (*a*)) → (tc-to-bv (0, size (*a*)) ≠ BTM)

THEOREM: tc-interpretation-of-inc-bit-vector
(bitvp (*a*) ∧ (*a* ≠ BTM))
→ (bv-adder-output (**t**, *a*, nat-to-bv (0, size (*a*)))
= tc-to-bv (**if** tc-in-rangep (add (bv-to-tc (*a*), 1), size (*a*))
then add (bv-to-tc (*a*), 1)
elseif negativep (add (bv-to-tc (*a*), 1))
then add (bv-to-tc (*a*), add (1, exp (2, size (*a*))))
else add (bv-to-tc (*a*),
add (1, − exp (2, size (*a*)))) **endif**,
size (*a*)))

EVENT: Disable tc-interpretation-of-inc-bit-vector.

THEOREM: tc-interpretation-of-dec-bit-vector
(bitvp (*a*) ∧ (*a* ≠ BTM))
→ (bv-subtracter-output (**t**, nat-to-bv (0, size (*a*)), *a*)
= tc-to-bv (**if** tc-in-rangep (add (bv-to-tc (*a*), -1), size (*a*))
then add (bv-to-tc (*a*), -1)
elseif negativep (add (bv-to-tc (*a*), -1))
then add (bv-to-tc (*a*), add (-1, exp (2, size (*a*))))
else add (bv-to-tc (*a*),
add (-1, − exp (2, size (*a*)))) **endif**,
size (*a*)))

EVENT: Disable tc-interpretation-of-dec-bit-vector.

THEOREM: tc-interpretation-of-inc-overflowp
(bitvp (a) ∧ $(a \neq$ BTM))
→ (bv-adder-overflowp (**t**, a, nat-to-bv (0, size (a)))
= (¬ tc-in-rangep (add (bv-to-tc (a), 1), size (a))))

EVENT: Disable tc-interpretation-of-inc-overflowp.

THEOREM: tc-interpretation-of-dec-overflowp
(bitvp (a) ∧ $(a \neq$ BTM))
→ (bv-subtracter-overflowp (**t**, nat-to-bv (0, size (a)), a)
= (¬ tc-in-rangep (add (bv-to-tc (a), -1), size (a))))

EVENT: Disable tc-interpretation-of-dec-overflowp.

THEOREM: tc-interpretation-of-negation-output
(bitvp (a) ∧ $(a \neq$ BTM))
→ (bv-subtracter-output (**f**, a, nat-to-bv (0, size (a)))
= tc-to-bv (**if** tc-in-rangep (tc-minus (bv-to-tc (a)), size (a))
then tc-minus (bv-to-tc (a))
elseif negativep (tc-minus (bv-to-tc (a)))
then add (exp (2, size (a)), tc-minus (bv-to-tc (a)))
else add (− exp (2, size (a)),
tc-minus (bv-to-tc (a))) **endif**,
size (a)))

EVENT: Disable tc-interpretation-of-negation-output.

THEOREM: tc-interpretation-of-negation-overflowp
(bitvp (a) ∧ $(a \neq$ BTM))
→ (bv-subtracter-overflowp (**f**, a, nat-to-bv (0, size (a)))
= (¬ tc-in-rangep (tc-minus (bv-to-tc (a)), size (a))))

EVENT: Disable tc-interpretation-of-negation-overflowp.

EVENT: Disable nat-of-zero-is-tc-of-zero.

EVENT: Disable tc-fix-of-add.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                                  ;;
;;            Two's complement operation of ALU                                     ;;
;;                                                                                  ;;
```

;;

Theorem: alu-correct-twos-complement

```
(bitvp (a)
 ∧  (a ≠ BTM)
 ∧  bitvp (b)
 ∧  (b ≠ BTM)
 ∧  (size (a) = size (b))
 ∧  boolp (c))
→   ((bv (bv-alu-cv (a, b, c, nat-to-bv (9, 4)))
       =  tc-to-bv (mod2 (bv-to-tc (a)), size (a)))
     ∧  (v (bv-alu-cv (a, b, c, nat-to-bv (9, 4))) = f)
     ∧  (bv (bv-alu-cv (a, b, c, nat-to-bv (7, 4)))
          =  tc-to-bv (if tc-in-rangep (add (bv-to-tc (b),
                                             tc-minus (bv-to-tc (a))),
                                        size (a))
                       then add (bv-to-tc (b), tc-minus (bv-to-tc (a)))
                       elseif negativep (add (bv-to-tc (b),
                                              tc-minus (bv-to-tc (a))))
                       then add (bv-to-tc (b),
                                 add (exp (2, size (a)),
                                      tc-minus (bv-to-tc (a))))
                       else add (bv-to-tc (b),
                                 add (− exp (2, size (a)),
                                      tc-minus (bv-to-tc (a)))) endif,
                       size (a)))
     ∧  (v (bv-alu-cv (a, b, c, nat-to-bv (7, 4)))
          =  (¬ tc-in-rangep (add (bv-to-tc (b),
                                   tc-minus (bv-to-tc (a))),
                              size (a))))
     ∧  (bv (bv-alu-cv (a, b, c, nat-to-bv (6, 4)))
          =  tc-to-bv (if tc-in-rangep (add (bv-to-tc (b),
                                             tc-minus (add (bv-to-tc (a),
                                                            carry (c)))),
                                        size (a))
                       then add (bv-to-tc (b),
                                 tc-minus (add (bv-to-tc (a), carry (c))))
                       elseif negativep (add (bv-to-tc (b),
                                              tc-minus (add (bv-to-tc (a),
                                                             carry (c)))))
                       then add (bv-to-tc (b),
                                 add (exp (2, size (a)),
```

```
                                        tc-minus (add (bv-to-tc (a),
                                                       carry (c)))))
                    else add (bv-to-tc (b),
                              add (− exp (2, size (a)),
                                   tc-minus (add (bv-to-tc (a),
                                                  carry (c)))) endif,
                    size (a)))
∧   (v (bv-alu-cv (a, b, c, nat-to-bv (6, 4)))
    =   (¬ tc-in-rangep (add (bv-to-tc (b),
                              tc-minus (add (bv-to-tc (a),
                                             carry (c)))),
                         size (a))))
∧   (bv (bv-alu-cv (a, b, c, nat-to-bv (5, 4)))
    =   tc-to-bv (if tc-in-rangep (add (bv-to-tc (a), -1), size (a))
                    then add (bv-to-tc (a), -1)
                    elseif negativep (add (bv-to-tc (a), -1))
                    then add (bv-to-tc (a), add (-1, exp (2, size (a))))
                    else add (bv-to-tc (a),
                              add (-1, − exp (2, size (a)))) endif,
                    size (a)))
∧   (v (bv-alu-cv (a, b, c, nat-to-bv (5, 4)))
    =   (¬ tc-in-rangep (add (bv-to-tc (a), -1), size (a))))
∧   (bv (bv-alu-cv (a, b, c, nat-to-bv (4, 4)))
    =   tc-to-bv (if tc-in-rangep (tc-minus (bv-to-tc (a)),
                                   size (a))
                    then tc-minus (bv-to-tc (a))
                    elseif negativep (tc-minus (bv-to-tc (a)))
                    then add (exp (2, size (a)),
                              tc-minus (bv-to-tc (a)))
                    else add (− exp (2, size (a)),
                              tc-minus (bv-to-tc (a))) endif,
                    size (a)))
∧   (v (bv-alu-cv (a, b, c, nat-to-bv (4, 4)))
    =   (¬ tc-in-rangep (tc-minus (bv-to-tc (a)), size (a))))
∧   (bv (bv-alu-cv (a, b, c, nat-to-bv (3, 4)))
    =   tc-to-bv (if tc-in-rangep (add (bv-to-tc (a), bv-to-tc (b)),
                                   size (a))
                    then add (bv-to-tc (a), bv-to-tc (b))
                    elseif negativep (add (bv-to-tc (a), bv-to-tc (b)))
                    then add (bv-to-tc (a),
                              add (bv-to-tc (b), exp (2, size (a))))
                    else add (bv-to-tc (a),
                              add (bv-to-tc (b),
                                   − exp (2, size (a)))) endif,
```

```
                    size (a)))
  ∧  (v (bv-alu-cv (a, b, c, nat-to-bv (3, 4)))
     =  (¬ tc-in-rangep (add (bv-to-tc (a), bv-to-tc (b)),
                             size (a))))
  ∧  (bv (bv-alu-cv (a, b, c, nat-to-bv (2, 4)))
     =  tc-to-bv (if tc-in-rangep (add (bv-to-tc (a),
                                        add (bv-to-tc (b), carry (c))),
                                   size (a))
                 then add (bv-to-tc (a),
                           add (bv-to-tc (b), carry (c)))
                 elseif negativep (add (bv-to-tc (a),
                                        add (bv-to-tc (b),
                                             carry (c))))
                 then add (bv-to-tc (a),
                           add (bv-to-tc (b),
                                add (carry (c), exp (2, size (a)))))
                 else add (bv-to-tc (a),
                           add (bv-to-tc (b),
                                add (carry (c),
                                     − exp (2, size (a))))) endif,
                 size (a)))
  ∧  (v (bv-alu-cv (a, b, c, nat-to-bv (2, 4)))
     =  (¬ tc-in-rangep (add (bv-to-tc (a),
                              add (bv-to-tc (b), carry (c))),
                         size (a))))
  ∧  (bv (bv-alu-cv (a, b, c, nat-to-bv (1, 4)))
     =  tc-to-bv (if tc-in-rangep (add (bv-to-tc (a), 1), size (a))
                 then add (bv-to-tc (a), 1)
                 elseif negativep (add (bv-to-tc (a), 1))
                 then add (bv-to-tc (a), add (1, exp (2, size (a))))
                 else add (bv-to-tc (a),
                           add (1, − exp (2, size (a)))) endif,
                 size (a)))
  ∧  (v (bv-alu-cv (a, b, c, nat-to-bv (1, 4)))
     =  (¬ tc-in-rangep (add (bv-to-tc (a), 1), size (a)))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;;          This file contains definitions and lemmas concerning the          ;;
;;          hardware necessary to build "big-machine".                        ;;
;;                                                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                                ;;
;;          Miscellaneous lemmas                                                  ;;
;;                                                                                ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: size-bv-bv-alu-cv
size (bv (bv-alu-cv (*a*, *b*, *c*, *op-code*))) = size (*a*)

EVENT: Disable bv-alu-cv.

EVENT: Disable bv-alu-cv-with-ifs.

DEFINITION:
fix-bool (*x*)
= **if** *x* **then t**
else f endif

DEFINITION:
properp (*x*)
= **if** *x* ≃ **nil then** *x* = **nil**
else properp (cdr (*x*)) **endif**

DEFINITION:
append (*x*, *y*)
= **if** *x* ≃ **nil then** *y*
else cons (car (*x*), append (cdr (*x*), *y*)) **endif**

THEOREM: assoc-of-append
append (append (*a*, *b*), *c*) = append (*a*, append (*b*, *c*))

THEOREM: length-0
(length (*x*) = 0) = (*x* ≃ **nil**)

DEFINITION:
v-nat-dec (*a*)
= **if** bv-to-nat (*a*) ≃ 0 **then** nat-to-bv (exp (2, size (*a*)) − 1, size (*a*))
else nat-to-bv (bv-to-nat (*a*) − 1, size (*a*)) **endif**

```
; around is received.
```

DEFINITION: v-nat-inc (a) = nat-to-bv $(1 + \text{bv-to-nat}(a), \text{size}(a))$

THEOREM: size-v-nat-dec
size (v-nat-dec (x)) = size (x)

THEOREM: size-v-nat-inc
size (v-nat-inc (x)) = size (x)

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                           ;;
;;          Memory accessors                                                 ;;
;;                                                                           ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:
nth (n, lst)
= **if** $n \simeq 0$ **then** car (lst)
else nth $(n - 1, \text{cdr}(\mathit{lst}))$ **endif**

DEFINITION: v-nth $(\mathit{v\text{-}n}, \mathit{lst})$ = nth (bv-to-nat $(\mathit{v\text{-}n})$, lst)

DEFINITION:
update-v $(c, \mathit{cell}, \mathit{value})$
= **if** truep (c) **then** *value*
else *cell* **endif**

DEFINITION:
update-nth $(c, n, \mathit{lst}, \mathit{value})$
= **if** truep (c) $\wedge$ listp (lst)
then if $n \simeq 0$ **then** cons $(\mathit{value}, \text{cdr}(\mathit{lst}))$
else cons (car (lst), update-nth $(c, n - 1, \text{cdr}(\mathit{lst}), \mathit{value})$) **endif**
else *lst* **endif**

DEFINITION:
update-v-nth $(c, \mathit{v\text{-}n}, \mathit{lst}, \mathit{value})$ = update-nth $(c, \text{bv-to-nat}(\mathit{v\text{-}n}), \mathit{lst}, \mathit{value})$

THEOREM: length-update-nth
length (update-nth $(c, n, \mathit{lst}, \mathit{value})$) = length (lst)

THEOREM: length-update-v-nth
length (update-v-nth $(c, n, \mathit{lst}, \mathit{value})$) = length (lst)

THEOREM: listp-update-nth
listp (update-nth $(c, n, \mathit{lst}, \mathit{value})$) = listp (lst)

THEOREM: listp-update-v-nth
listp (update-v-nth (c, n, *lst*, *value*)) = listp (*lst*)

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                              ;;
;;          Zero and Negative flag interpretation lemmas                        ;;
;;                                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:
b-bv-nzerop (a)
= **if** bitvp (a)
  then if a = BTM **then f**
    else b-or (bit (a), b-bv-nzerop (vec (a))) **endif**
  else f endif

DEFINITION: b-bv-zerop (a) = b-not (b-bv-nzerop (a))

DEFINITION:
v-zerop (a)
= **if** bitvp (a)
  then if a = BTM **then t**
    else (bit (a) = **f**) $\wedge$ v-zerop (vec (a)) **endif**
  else t endif

THEOREM: b-bv-zerop-equal-v-zerop
b-bv-zerop (a) = v-zerop (a)

```
;  The following are the natural number and tc intepretation lemmas for
;  the zero flag.
```

THEOREM: v-zerop-equal-bv-to-nat-zero
v-zerop (a) = (bv-to-nat (a) $\simeq$ 0)

THEOREM: v-zerop-equal-bv-to-tc-zero
v-zerop (a) = (0 = bv-to-tc (a))

EVENT: Disable v-zerop-equal-bv-to-nat-zero.

EVENT: Disable v-zerop-equal-bv-to-tc-zero.

```
;  The following is the tc interpretation lemma for the negative flag.
```

THEOREM: n-flag-equals-negativep
bitn $(a, \text{size}(a)) = \text{negativep}(\text{bv-to-tc}(a))$

EVENT: Disable n-flag-equals-negativep.

DEFINITION:
bv-equal (a, b)
= **if** bitvp (a)
then if $a = \text{BTM}$ **then t**
else b-and (b-equv (bit (a), bit (b)), bv-equal (vec (a), vec (b))) **endif**
else f endif

THEOREM: bv-equal-is-equal
$(\text{bitvp}(a) \land \text{bitvp}(b) \land (\text{size}(a) = \text{size}(b)))$
$\rightarrow$ $(\text{bv-equal}(a, b) = (a = b))$

EVENT: Disable bv-equal-is-equal.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                             ;;
;;          Assumptions made about the various registers                      ;;
;;                                                                             ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION: NXSZ = 4

```
; width of micro address reg
```

DEFINITION: MACHINE-SIZE = 16

```
; Word width of micro-processor
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                             ;;
;;          Register, RAMP, and ROMP recognizer                                ;;
;;                                                                             ;;
;;          A ram and rom checker are presented below and is used to ensure    ;;
;;          that RAM and ROM (constant function) have certain properties.      ;;
;;                                                                             ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION: sizep (*register*, *width*) = (size (*register*) = *width*)

DEFINITION:
every-member-sizep (*lst*, *n*)
= **if** *lst* ≃ **nil then t**
else sizep (car (*lst*), *n*) ∧ every-member-sizep (cdr (*lst*), *n*) **endif**

DEFINITION:
ramp (*ram*, *width*, *locations*)
= ((length (*ram*) = *locations*) ∧ every-member-sizep (*ram*, *width*))

DEFINITION:
romp (*rom*, *width*, *locations*)
= ((length (*rom*) = *locations*) ∧ every-member-sizep (*rom*, *width*))

THEOREM: every-member-sizep-implies-size-nth
((*n* < length (*lst*)) ∧ every-member-sizep (*lst*, *width*))
→ (size (nth (*n*, *lst*)) = *width*)

THEOREM: every-member-sizep-implies-size-nth-machine-size
((*n* < length (*lst*)) ∧ every-member-sizep (*lst*, MACHINE-SIZE))
→ (size (nth (*n*, *lst*)) = MACHINE-SIZE)

THEOREM: every-member-sizep-implies-size-nth-0
(every-member-sizep (*lst*, *width*) ∧ (0 < length (*lst*)))
→ (size (car (*lst*)) = *width*)

EVENT: Disable boolp.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                             ;;
;;          Op-code accessors                                                  ;;
;;                                                                             ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:
bv-op-code (*i-reg*)
= bitv (bitn (*i-reg*, `13`),
bitv (bitn (*i-reg*, `14`),
bitv (bitn (*i-reg*, `15`), bitv (bitn (*i-reg*, `16`), BTM))))

DEFINITION: b-move-op (*i-reg*) = bitn (*i-reg*, `12`)

DEFINITION: b-cc-set (*i-reg*) = bitn (*i-reg*, `11`)

DEFINITION:
b-direct-reg-b (*i-reg*) = b-nor (bitn (*i-reg*, `9`), bitn (*i-reg*, `10`))

DEFINITION:
b-indirect-reg-b (*i-reg*) = b-and (bitn (*i-reg*, `9`), b-not (bitn (*i-reg*, `10`)))

DEFINITION:
b-indirect-reg-b-dec (*i-reg*) = b-and (b-not (bitn (*i-reg*, `9`)), bitn (*i-reg*, `10`))

DEFINITION:
b-indirect-reg-b-inc (*i-reg*) = b-and (bitn (*i-reg*, `9`), bitn (*i-reg*, `10`))

DEFINITION:
bv-oprd-b (*i-reg*)
= bitv (bitn (*i-reg*, `6`), bitv (bitn (*i-reg*, `7`), bitv (bitn (*i-reg*, `8`), BTM)))

DEFINITION:
b-direct-reg-a (*i-reg*) = b-nor (bitn (*i-reg*, `4`), bitn (*i-reg*, `5`))

DEFINITION:
b-indirect-reg-a (*i-reg*) = b-and (bitn (*i-reg*, `4`), b-not (bitn (*i-reg*, `5`)))

DEFINITION:
b-indirect-reg-a-dec (*i-reg*) = b-and (b-not (bitn (*i-reg*, `4`)), bitn (*i-reg*, `5`))

DEFINITION:
b-indirect-reg-a-inc (*i-reg*) = b-and (bitn (*i-reg*, `4`), bitn (*i-reg*, `5`))

DEFINITION:
bv-oprd-a (*i-reg*)
= bitv (bitn (*i-reg*, `1`), bitv (bitn (*i-reg*, `2`), bitv (bitn (*i-reg*, `3`), BTM)))

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                          ;;
;;          Sub-functions for FM8501                                        ;;
;;                                                                          ;;
;;          NOTE:  Many definitions below are presented in pairs.  A hardware   ;;
```

```
;;                 version is presented along with a version that is more       ;;
;;                 agreeable to the theorem-prover.  The second of these        ;;
;;                 paired definitions is has a "WITH-IFS" ending.  Not all      ;;
;;                 of the hardware functions have a "WITH-IFS" pair.            ;;
;;                                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:
bv-alu-op-code (*i-reg*)
= bv-if (b-move-op (*i-reg*), nat-to-bv (0, 4), bv-op-code (*i-reg*))

DEFINITION:
b-store-alu-result (*c-flag*, *v-flag*, *z-flag*, *n-flag*, *i-reg*)
= b-or (b-or (b-not (b-move-op (*i-reg*)), bitn (bv-op-code (*i-reg*), 4)),
b-or (b-or (b-or (b-and (b-not (*c-flag*),
bv-equal (bv-op-code (*i-reg*), nat-to-bv (0, 4))),
b-and (*c-flag*,
bv-equal (bv-op-code (*i-reg*), nat-to-bv (1, 4)))),
b-or (b-and (b-not (*v-flag*),
bv-equal (bv-op-code (*i-reg*), nat-to-bv (2, 4))),
b-and (*v-flag*,
bv-equal (bv-op-code (*i-reg*), nat-to-bv (3, 4))))),
b-or (b-or (b-and (b-not (*z-flag*),
bv-equal (bv-op-code (*i-reg*), nat-to-bv (4, 4))),
b-and (*z-flag*,
bv-equal (bv-op-code (*i-reg*), nat-to-bv (5, 4)))),
b-or (b-and (b-not (*n-flag*),
bv-equal (bv-op-code (*i-reg*), nat-to-bv (6, 4))),
b-and (*n-flag*,
bv-equal (bv-op-code (*i-reg*), nat-to-bv (7, 4)))))))

DEFINITION:
b-store-alu-result-with-ifs (*c-flag*, *v-flag*, *z-flag*, *n-flag*, *i-reg*)
= ((¬ b-move-op (*i-reg*))
∨ bitn (bv-op-code (*i-reg*), 4)
∨ (bv-op-code (*i-reg*)
= nat-to-bv (**if** *c-flag* **then** 1
else 0 **endif**,
4))
∨ (bv-op-code (*i-reg*)
= nat-to-bv (**if** *v-flag* **then** 3
else 2 **endif**,
4))

    ∨ (bv-op-code (*i-reg*)
        = nat-to-bv (**if** *z-flag* **then** 5
                      else 4 **endif**,
                      4))
    ∨ (bv-op-code (*i-reg*)
        = nat-to-bv (**if** *n-flag* **then** 7
                      else 6 **endif**,
                      4)))

THEOREM: b-store-alu-result-ifs
b-store-alu-result (*c-flag*, *v-flag*, *z-flag*, *n-flag*, *i-reg*)
= b-store-alu-result-with-ifs (*c-flag*, *v-flag*, *z-flag*, *n-flag*, *i-reg*)

EVENT: Disable b-store-alu-result.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;;          Micro-rom                                                         ;;
;;                                                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:
make-micro-word (*l*)
= **if** $l \simeq$ **nil** **then** BTM
  elseif cdr (l) $\simeq$ **nil** **then** nat-to-bv (car (l), NXSZ)
  else bitv (**if** car (l) = 't **then** **t**
              else **f** **endif**,
              make-micro-word (cdr (l))) **endif**

DEFINITION:
make-micro-rom (*l*)
= **if** $l \simeq$ **nil** **then** **nil**
  else cons (make-micro-word (car (l)), make-micro-rom (cdr (l))) **endif**

DEFINITION:
MICRO-STORE
= make-micro-rom ('((f f f f f f f f f t f f f f f f 1)
                   (f t f t f t f f f t f t f f f f 2)
                   (f f f t f f f f t f f f f f f f 3)
                   (f f f f f f f f f t f f f f t f 4)
                   (f f f f t f f t f t f t f f f f 5)
                   (f f f f t f f f t f f f f f f f 6)

```
(f f f f t f f f f t t f f f f f 7)
(f f f f f f f t f t f t f f f f 8)
(f f f f f f f f t f f f f f f f 9)
(f f f f f f f f t f f f t f f f 10)
(f f f f f f f f t f f f t f f f 11)
(t f t f f f f f f f f f f f f t 12)
(f f f f t f t f f t f f f f f f 13)
(f f f f f f t f f t f f f f f f 1)
(f f f f f f f f f t f f f f f f 0)
(f f f f f f f f f t f f f f f f 0)))
```

DEFINITION: micro-rom $(addr)$ = v-nth $(addr,$ MICRO-STORE$)$

```
;  Accessors for data in the ROM named "micro-store".
```

DEFINITION: b-dout-incdec (x) = bitn $(x,$ 1$)$

```
; Data out reg or address unit
```

DEFINITION: b-force-inc (x) = bitn $(x,$ 2$)$

```
; Force increment for address
```

DEFINITION: b-en-no-store (x) = bitn $(x,$ 3$)$

```
; Let "NO-STORE" effect sequencer
```

DEFINITION: b-ir-mem-ref (x) = bitn $(x,$ 4$)$

```
; Memory reference for instruction
```

DEFINITION: b-oprda-oprdb (x) = bitn $(x,$ 5$)$

```
; Select operand A or operand B
```

DEFINITION: b-pc-regnum (x) = bitn $(x,$ 6$)$

```
; Select PC or register in file
```

DEFINITION: b-postinc (x) = bitn $(x,$ 7$)$

```
; Post-increment address mode
```

DEFINITION: b-predec (x) = bitn $(x,$ 8$)$

```
; Pre-decrement address mode
```

DEFINITION: b-rd (x) = bitn $(x,$ 9$)$

```
; Read cycle
```

DEFINITION: b-seq (x) = bitn $(x,$ 10$)$

```
; Sequence micro-address register
```

DEFINITION: b-we-a-reg (x) = bitn $(x,$ 11$)$

```
; Write enable A register
```

DEFINITION: b-we-addr-out (x) = bitn $(x,$ 12$)$

```
; Write enable address-out register
```

DEFINITION: b-we-b-reg (x) = bitn $(x,$ 13$)$

```
; Write enable B register
```

DEFINITION: b-we-alu-result (x) = bitn $(x,$ 14$)$

```
; WE DATA-OUT, CC-REG, NO-STORE
```

DEFINITION: b-we-ir (x) = bitn $(x,$ 15$)$

```
; Write enable instruction register
```

DEFINITION: b-wr (x) = bitn $(x,$ 16$)$

```
; Write cycle
```

DEFINITION:
bv-next (x)
= bitv (bitn $(x,$ 17$)$,
bitv (bitn $(x,$ 18$)$, bitv (bitn $(x,$ 19$)$, bitv (bitn $(x,$ 20$)$, BTM))))

DEFINITION:
right-bv-next-size (lst)
= **if** $lst \simeq$ **nil then t**
else (size (bv-next (car (lst))) = NXSZ)
$\wedge$ right-bv-next-size (cdr (lst)) **endif**

THEOREM: size-bv-next-micro-rom-lemma
size (bv-next (x)) = 4

EVENT: Disable micro-rom.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;;         Hardware variable and constant assumptions for FM8501.             ;;
;;                                                                            ;;
;;      The following definition describes the assumptions about the various  ;;
;;      registers (variables) and the ROM used in the construction of the     ;;
;;      FM8501 microprocessor.                                                ;;
;;                                                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:

standard-hyps (*mar*,
read,
write,
dtack,
reset,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem)

= (romp (MICRO-STORE, 20, exp (2, NXSZ))
∧ sizep (*mar*, NXSZ)
∧ boolp (*read*)
∧ boolp (*write*)
∧ boolp (*dtack*)
∧ boolp (*reset*)
∧ boolp (*no-store*)
∧ sizep (*data-out*, MACHINE-SIZE)
∧ ramp (*reg-file*, MACHINE-SIZE, 8)
∧ sizep (*addr-out*, MACHINE-SIZE)
∧ boolp (*c-flag*)
∧ boolp (*v-flag*)
∧ boolp (*z-flag*)

$\wedge$ boolp (*n-flag*)
$\wedge$ sizep (*a-reg*, MACHINE-SIZE)
$\wedge$ sizep (*b-reg*, MACHINE-SIZE)
$\wedge$ sizep (*i-reg*, MACHINE-SIZE)
$\wedge$ sizep (*visual-mem*, MACHINE-SIZE)
$\wedge$ ramp (*real-mem*, MACHINE-SIZE, exp (2, 16)))

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;;          Miscellanous lemmas                                               ;;
;;                                                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: every-member-sizep-implies-bitvp
(every-member-sizep (*lst*, *width*) $\wedge$ ($n <$ length (*lst*)) $\wedge$ (*width* $\not\simeq$ 0))
$\rightarrow$ bitvp (nth (n, *lst*))

THEOREM: every-member-sizep-implies-bitvp-machine-size
(every-member-sizep (*lst*, MACHINE-SIZE) $\wedge$ ($n <$ length (*lst*)))
$\rightarrow$ bitvp (nth (n, *lst*))

THEOREM: sizep-implies-bitvp
(size (a) $=$ ($1 + n$)) $\rightarrow$ bitvp (a)

```
;  The following lemma subsumes trunc-size.
```

THEOREM: trunc-size-bridge
(bitvp (a) $\wedge$ ($n =$ size (a))) $\rightarrow$ (trunc (a, n) $= a$)

THEOREM: b-if-works
(boolp (c) $\wedge$ boolp (a) $\wedge$ boolp (b))
$\rightarrow$ (b-if (c, a, b)
  $=$ **if** c **then** a
  else b **endif**)

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;;          Functions for composing the micro-processor                       ;;
;;                                                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:
bv-reg-select (*i-reg*, *mar*, *reset*)
= bv-if (b-or (*reset*, b-pc-regnum (micro-rom (*mar*))),
nat-to-bv (0, 3),
bv-if (b-oprda-oprdb (micro-rom (*mar*)),
bv-oprd-a (*i-reg*),
bv-oprd-b (*i-reg*)))

DEFINITION:
bv-reg-select-with-ifs (*i-reg*, *mar*, *reset*)
= **if** b-or (*reset*, b-pc-regnum (micro-rom (*mar*))) **then** nat-to-bv (0, 3)
elseif b-oprda-oprdb (micro-rom (*mar*)) **then** bv-oprd-a (*i-reg*)
else bv-oprd-b (*i-reg*) **endif**

THEOREM: bv-reg-select-ifs
bv-reg-select (*i-reg*, *mar*, *reset*) = bv-reg-select-with-ifs (*i-reg*, *mar*, *reset*)

DEFINITION:
b-oprd-mem-ref (*mar*, *i-reg*)
= b-if (b-oprda-oprdb (micro-rom (*mar*)),
b-not (b-direct-reg-a (*i-reg*)),
b-not (b-direct-reg-b (*i-reg*)))

DEFINITION:
b-oprd-mem-ref-with-ifs (*mar*, *i-reg*)
= **if** b-oprda-oprdb (micro-rom (*mar*)) **then** b-not (b-direct-reg-a (*i-reg*))
else b-not (b-direct-reg-b (*i-reg*)) **endif**

THEOREM: b-oprd-mem-ref-ifs
b-oprd-mem-ref (*mar*, *i-reg*) = b-oprd-mem-ref-with-ifs (*mar*, *i-reg*)

EVENT: Disable b-oprd-mem-ref.

```
; This disable was not here when FM8501 proved.

;  Next micro-address
```

DEFINITION:
mar (*mar*, *i-reg*, *dtack*, *reset*, *no-store*)
= update-v (b-or (b-or (*reset*, b-or (*dtack*, b-seq (micro-rom (*mar*)))),
b-or (b-and (b-en-no-store (micro-rom (*mar*)), *no-store*),
b-and (b-not (b-oprd-mem-ref (*mar*, *i-reg*)),
b-not (b-ir-mem-ref (micro-rom (*mar*)))))),
mar,
bv-if (*reset*, nat-to-bv (0, NXSZ), bv-next (micro-rom (*mar*))))

```
;  Read register
```

DEFINITION:
read (*mar*, *i-reg*)
= b-and (b-or (b-oprd-mem-ref (*mar*, *i-reg*), b-ir-mem-ref (micro-rom (*mar*))),
b-rd (micro-rom (*mar*)))

```
;  Write register
```

DEFINITION:
write (*mar*, *i-reg*, *no-store*)
= b-and (b-wr (micro-rom (*mar*)),
b-and (b-oprd-mem-ref (*mar*, *i-reg*), b-not (*no-store*)))

```
;  Data acknowledgement
```

DEFINITION:
dtack (*current-oracle-entry*) = fix-bool (car (*current-oracle-entry*))

```
;  Reset function
```

DEFINITION:
reset (*current-oracle-entry*) = fix-bool (cadr (*current-oracle-entry*))

```
;  No-store -- defines the flag which allows FM8501 to do conditional stores.
```

DEFINITION:
no-store (*no-store*, *c-flag*, *v-flag*, *z-flag*, *n-flag*, *i-reg*, *mar*)
= update-v (b-we-alu-result (micro-rom (*mar*)),
no-store,
b-not (b-store-alu-result (*c-flag*, *v-flag*, *z-flag*, *n-flag*, *i-reg*)))

DEFINITION:
no-store-with-ifs (*no-store*, *c-flag*, *v-flag*, *z-flag*, *n-flag*, *i-reg*, *mar*)
= **if** b-we-alu-result (micro-rom (*mar*))
then ¬ b-store-alu-result-with-ifs (*c-flag*,
v-flag,
z-flag,
n-flag,
i-reg)
else *no-store* **endif**

THEOREM: no-store-ifs
no-store (*no-store*, *c-flag*, *v-flag*, *z-flag*, *n-flag*, *i-reg*, *mar*)
= no-store-with-ifs (*no-store*, *c-flag*, *v-flag*, *z-flag*, *n-flag*, *i-reg*, *mar*)

EVENT: Disable no-store.

```
;  Bit-vector result of the ALU
```

DEFINITION:
data-out (*data-out*, *a-reg*, *b-reg*, *c-flag*, *i-reg*, *mar*)
= update-v (b-we-alu-result (micro-rom (*mar*)),
data-out,
bv (bv-alu-cv (*a-reg*, *b-reg*, *c-flag*, bv-alu-op-code (*i-reg*))))

DEFINITION:
data-out-with-ifs (*data-out*, *a-reg*, *b-reg*, *c-flag*, *i-reg*, *mar*)
= **if** b-we-alu-result (micro-rom (*mar*))
then bv (bv-alu-cv (*a-reg*, *b-reg*, *c-flag*, bv-alu-op-code (*i-reg*)))
else *data-out* **endif**

THEOREM: data-out-ifs
data-out (*data-out*, *a-reg*, *b-reg*, *c-flag*, *i-reg*, *mar*)
= data-out-with-ifs (*data-out*, *a-reg*, *b-reg*, *c-flag*, *i-reg*, *mar*)

EVENT: Disable data-out.

```
;  Calculates the next value of the register file.
```

DEFINITION:
reg-file (*reg-file*, *data-out*, *i-reg*, *mar*, *no-store*, *reset*)
= update-v-nth (b-or (b-or (*reset*, b-force-inc (micro-rom (*mar*))),
b-or (b-or (b-pc-regnum (micro-rom (*mar*)),
b-and (b-wr (micro-rom (*mar*)),
b-and (b-not (*no-store*),
b-direct-reg-b (*i-reg*)))),
b-or (b-and (b-predec (micro-rom (*mar*)),
b-if (b-oprda-oprdb (micro-rom (*mar*)),
b-indirect-reg-a-dec (*i-reg*),
b-indirect-reg-b-dec (*i-reg*))),
b-and (b-postinc (micro-rom (*mar*)),
b-if (b-oprda-oprdb (micro-rom (*mar*)),
b-indirect-reg-a-inc (*i-reg*),

```
                                                b-indirect-reg-b-inc (i-reg))))))),
                        bv-reg-select (i-reg, mar, reset),
                        reg-file,
                        bv-if (b-or (b-dout-incdec (micro-rom (mar)), reset),
                                bv-if (reset, nat-to-bv (0, MACHINE-SIZE), data-out),
                                bv-if (b-or (b-force-inc (micro-rom (mar)),
                                             b-if (b-oprda-oprdb (micro-rom (mar)),
                                                  b-indirect-reg-a-inc (i-reg),
                                                  b-indirect-reg-b-inc (i-reg))),
                                       bv-adder-output (t,
                                                        nat-to-bv (0, MACHINE-SIZE),
                                                        v-nth (bv-reg-select (i-reg,
                                                                              mar,
                                                                              reset),
                                                               reg-file)),
                                       bv-subtracter-output (t,
                                                             nat-to-bv (0,
                                                                        MACHINE-SIZE),
                                                             v-nth (bv-reg-select (i-reg,
                                                                                   mar,
                                                                                   reset),
                                                                    reg-file)))))

;  Defines the output address presented by FM8501.


DEFINITION:
addr-out (addr-out, reg-file, i-reg, mar, reset)
=   update-v (b-we-addr-out (micro-rom (mar)),
                addr-out,
                bv-if (b-and (b-predec (micro-rom (mar)),
                              b-if (b-oprda-oprdb (micro-rom (mar)),
                                   b-indirect-reg-a-dec (i-reg),
                                   b-indirect-reg-b-dec (i-reg))),
                       bv-subtracter-output (t,
                                             nat-to-bv (0, MACHINE-SIZE),
                                             v-nth (bv-reg-select (i-reg,
                                                                   mar,
                                                                   reset),
                                                    reg-file)),
                       v-nth (bv-reg-select (i-reg, mar, reset), reg-file)))

;  Carry flag
```

DEFINITION:
c-flag (*c-flag*, *a-reg*, *b-reg*, *i-reg*, *mar*)
= update-v (b-and (b-we-alu-result (micro-rom (*mar*)), b-cc-set (*i-reg*)),
c-flag,
c (bv-alu-cv (*a-reg*, *b-reg*, *c-flag*, bv-alu-op-code (*i-reg*))))

DEFINITION:
c-flag-with-ifs (*c-flag*, *a-reg*, *b-reg*, *i-reg*, *mar*)
= **if** b-and (b-we-alu-result (micro-rom (*mar*)), b-cc-set (*i-reg*))
then c (bv-alu-cv (*a-reg*, *b-reg*, *c-flag*, bv-alu-op-code (*i-reg*)))
else *c-flag* **endif**

THEOREM: c-flag-ifs
c-flag (*c-flag*, *a-reg*, *b-reg*, *i-reg*, *mar*)
= c-flag-with-ifs (*c-flag*, *a-reg*, *b-reg*, *i-reg*, *mar*)

EVENT: Disable c-flag.

```
;  The overflow flag
```

DEFINITION:
v-flag (*v-flag*, *a-reg*, *b-reg*, *c-flag*, *i-reg*, *mar*)
= update-v (b-and (b-we-alu-result (micro-rom (*mar*)), b-cc-set (*i-reg*)),
v-flag,
v (bv-alu-cv (*a-reg*, *b-reg*, *c-flag*, bv-alu-op-code (*i-reg*))))

DEFINITION:
v-flag-with-ifs (*v-flag*, *a-reg*, *b-reg*, *c-flag*, *i-reg*, *mar*)
= **if** b-and (b-we-alu-result (micro-rom (*mar*)), b-cc-set (*i-reg*))
then v (bv-alu-cv (*a-reg*, *b-reg*, *c-flag*, bv-alu-op-code (*i-reg*)))
else *v-flag* **endif**

THEOREM: v-flag-ifs
v-flag (*v-flag*, *a-reg*, *b-reg*, *c-flag*, *i-reg*, *mar*)
= v-flag-with-ifs (*v-flag*, *a-reg*, *b-reg*, *c-flag*, *i-reg*, *mar*)

EVENT: Disable v-flag.

```
;  The zero flag
```

DEFINITION:
z-flag (*z-flag*, *a-reg*, *b-reg*, *c-flag*, *i-reg*, *mar*)

= update-v (b-and (b-we-alu-result (micro-rom (*mar*)), b-cc-set (*i-reg*)),
z-flag,
b-bv-zerop (bv (bv-alu-cv (*a-reg*,
b-reg,
c-flag,
bv-alu-op-code (*i-reg*)))))

DEFINITION:
z-flag-with-ifs (*z-flag*, *a-reg*, *b-reg*, *c-flag*, *i-reg*, *mar*)
= **if** b-and (b-we-alu-result (micro-rom (*mar*)), b-cc-set (*i-reg*))
then b-bv-zerop (bv (bv-alu-cv (*a-reg*, *b-reg*, *c-flag*, bv-alu-op-code (*i-reg*))))
else *z-flag* **endif**

THEOREM: z-flag-ifs
z-flag (*z-flag*, *a-reg*, *b-reg*, *c-flag*, *i-reg*, *mar*)
= z-flag-with-ifs (*z-flag*, *a-reg*, *b-reg*, *c-flag*, *i-reg*, *mar*)

EVENT: Disable z-flag.

```
;  Two's complement negative flag
```

DEFINITION:
n-flag (*n-flag*, *a-reg*, *b-reg*, *c-flag*, *i-reg*, *mar*)
= update-v (b-and (b-we-alu-result (micro-rom (*mar*)), b-cc-set (*i-reg*)),
n-flag,
bitn (bv (bv-alu-cv (*a-reg*, *b-reg*, *c-flag*, bv-alu-op-code (*i-reg*))),
MACHINE-SIZE))

DEFINITION:
n-flag-with-ifs (*n-flag*, *a-reg*, *b-reg*, *c-flag*, *i-reg*, *mar*)
= **if** b-and (b-we-alu-result (micro-rom (*mar*)), b-cc-set (*i-reg*))
then bitn (bv (bv-alu-cv (*a-reg*, *b-reg*, *c-flag*, bv-alu-op-code (*i-reg*))),
MACHINE-SIZE)
else *n-flag* **endif**

THEOREM: n-flag-ifs
n-flag (*n-flag*, *a-reg*, *b-reg*, *c-flag*, *i-reg*, *mar*)
= n-flag-with-ifs (*n-flag*, *a-reg*, *b-reg*, *c-flag*, *i-reg*, *mar*)

EVENT: Disable n-flag.

```
;  Operand A for the ALU
```

DEFINITION:
a-reg (*a-reg*, *visual-mem*, *reg-file*, *i-reg*, *mar*, *reset*)
= update-v (b-we-a-reg (micro-rom (*mar*)),
  a-reg,
  bv-if (b-direct-reg-a (*i-reg*),
    v-nth (bv-reg-select (*i-reg*, *mar*, *reset*), *reg-file*),
    visual-mem))

```
;  Operand B for the ALU
```

DEFINITION:
b-reg (*b-reg*, *visual-mem*, *reg-file*, *i-reg*, *mar*, *reset*)
= update-v (b-we-b-reg (micro-rom (*mar*)),
  b-reg,
  bv-if (b-direct-reg-b (*i-reg*),
    v-nth (bv-reg-select (*i-reg*, *mar*, *reset*), *reg-file*),
    visual-mem))

```
;  Fills the instruction register
```

DEFINITION:
i-reg (*i-reg*, *visual-mem*, *mar*)
= update-v (b-we-ir (micro-rom (*mar*)), *i-reg*, *visual-mem*)

DEFINITION:
i-reg-with-ifs (*i-reg*, *visual-mem*, *mar*)
= **if** b-we-ir (micro-rom (*mar*)) **then** *visual-mem*
  else *i-reg* **endif**

THEOREM: i-reg-ifs
i-reg (*i-reg*, *visual-mem*, *mar*) = i-reg-with-ifs (*i-reg*, *visual-mem*, *mar*)

EVENT: Disable i-reg.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                              ;;
;;          Memory process                                                      ;;
;;                                                                              ;;
;;          The memory process is interesting as it requires history to         ;;
;;          operate.  VISUAL-MEM is the description of how a memory that        ;;
;;          FM8501 reads is expected to act.  A "correct" memory value is       ;;
;;          only returned after the READ line has been on for two clock "ticks",;;
;;          the WRITE line is off, DTACK is on, RESET is off, and DATA-OUT      ;;
```

```
;;          and ADDR-OUT have been stable for two clock "ticks".  VISUAL-MEM    ;;
;;          embodies the assumptions made about reading memory.  REAL-MEM      ;;
;;          is a function that actually updates the memory.  Its assumptions   ;;
;;          are expressed similarily as VISUAL-MEM.  The WATCH-DOG allows      ;;
;;          one tick's worth of state to be kept concerning READ, WRITE,       ;;
;;          DTACK, DATA-OUT, and ADDR-OUT.                                     ;;
;;                                                                             ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

EVENT: Add the shell *watch-dog*, with recognizer function symbol *watch-dogp* and 5 accessors: *read-1*, with type restriction (none-of) and default value false; *write-1*, with type restriction (none-of) and default value false; *dtack-1-oracle*, with type restriction (none-of) and default value false; *data-out-1*, with type restriction (one-of bitvp) and default value btm; *addr-out-1*, with type restriction (one-of bitvp) and default value btm.

EVENT: Introduce the function symbol *default-visual-mem-value* of 0 arguments.

```
; Default value from memory when memory
                                          ; has not been selected to return a value.
```

DEFINITION:
visual-mem (*real-mem*,
read,
write,
addr-out,
memory-watch-dog-history,
dtack-oracle,
reset-oracle)
= trunc (**if** (*read* ∧ read-1 (*memory-watch-dog-history*))
∧ ((¬ *write*)
∧ (¬ write-1 (*memory-watch-dog-history*)))
∧ (*addr-out* = addr-out-1 (*memory-watch-dog-history*))
∧ (*dtack-oracle* ∧ dtack-1-oracle (*memory-watch-dog-history*))
∧ (¬ *reset-oracle*) **then** v-nth (*addr-out*, *real-mem*)
else DEFAULT-VISUAL-MEM-VALUE **endif**,
MACHINE-SIZE)

DEFINITION:
real-mem (*real-mem*,

read,
write,
addr-out,
data-out,
memory-watch-dog-history,
dtack-oracle,
reset-oracle)

$=$ update-v-nth ((*write* $\wedge$ write-1 (*memory-watch-dog-history*))
$\wedge$ (($\neg$ *read*)
$\wedge$ ($\neg$ read-1 (*memory-watch-dog-history*)))
$\wedge$ (*addr-out* $=$ addr-out-1 (*memory-watch-dog-history*))
$\wedge$ (*dtack-oracle* $\wedge$ ($\neg$ dtack-1-oracle (*memory-watch-dog-history*)))
$\wedge$ ($\neg$ *reset-oracle*),
addr-out,
real-mem,
data-out)

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;;          "big-machine" -- the definition of the microprocessor             ;;
;;                                                                            ;;
;;          This function is the definition of the FM8501 microprocessor.     ;;
;;          BIG-MACHINE executes as along as there are clock "ticks".  Clock  ;;
;;          "ticks" are characterized by there being elements left in the     ;;
;;          ORACLE.  Otherwise, this function just stops, effectively         ;;
;;          freezing the value of its variables (registers).                  ;;
;;                                                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:
big-machine (*mar*,
read,
write,
dtack,
reset,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,

n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
memory-watch-dog-history,
oracle)

= **if** *oracle* $\simeq$ **nil**
then list (*mar*,
read,
write,
dtack,
reset,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
memory-watch-dog-history)
else big-machine (mar (*mar*, *i-reg*, *dtack*, *reset*, *no-store*),
read (*mar*, *i-reg*),
write (*mar*, *i-reg*, *no-store*),
dtack (car (*oracle*)),
reset (car (*oracle*)),
no-store (*no-store*,
c-flag,
v-flag,
z-flag,
n-flag,
i-reg,
mar),
data-out (*data-out*, *a-reg*, *b-reg*, *c-flag*, *i-reg*, *mar*),
reg-file (*reg-file*,
data-out,
i-reg,

mar,
no-store,
reset),
addr-out (*addr-out*, *reg-file*, *i-reg*, *mar*, *reset*),
c-flag (*c-flag*, *a-reg*, *b-reg*, *i-reg*, *mar*),
v-flag (*v-flag*, *a-reg*, *b-reg*, *c-flag*, *i-reg*, *mar*),
z-flag (*z-flag*, *a-reg*, *b-reg*, *c-flag*, *i-reg*, *mar*),
n-flag (*n-flag*, *a-reg*, *b-reg*, *c-flag*, *i-reg*, *mar*),
a-reg (*a-reg*,
visual-mem,
reg-file,
i-reg,
mar,
reset),
b-reg (*b-reg*,
visual-mem,
reg-file,
i-reg,
mar,
reset),
i-reg (*i-reg*, *visual-mem*, *mar*),
visual-mem (*real-mem*,
read,
write,
addr-out,
memory-watch-dog-history,
dtack (car (*oracle*)),
reset (car (*oracle*))),
real-mem (*real-mem*,
read,
write,
addr-out,
data-out,
memory-watch-dog-history,
dtack (car (*oracle*)),
reset (car (*oracle*))),
watch-dog (*read*,
write,
dtack (car (*oracle*)),
data-out,
addr-out),
cdr (*oracle*)) **endif**

;;

```
;;                                                                              ;;
;;          "big-machine" operation on lists and empty lists                    ;;
;;                                                                              ;;
;;          The following two lemmas describe what BIG-MACHINE does when        ;;
;;          clock "ticks" are available or the clock has been exhausted.        ;;
;;          This information is easily available by looking at the definition   ;;
;;          BIG-MACHINE; however, these formulations make it easier for the     ;;
;;          theorem-prover to reason about BIG-MACHINE.                         ;;
;;                                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: open-big-machine-on-nlistp
(¬ listp (*oracle*))
→ (big-machine (*mar*,
read,
write,
dtack,
reset,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
memory-watch-dog-history,
oracle)
= list (*mar*,
read,
write,
dtack,
reset,
no-store,
data-out,
reg-file,
addr-out,

c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
memory-watch-dog-history))

THEOREM: open-big-machine-on-listp
listp (*oracle*)
→ (big-machine (*mar*,
read,
write,
dtack,
reset,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
memory-watch-dog-history,
oracle)
= big-machine (mar (*mar*, *i-reg*, *dtack*, *reset*, *no-store*),
read (*mar*, *i-reg*),
write (*mar*, *i-reg*, *no-store*),
dtack (car (*oracle*)),
reset (car (*oracle*)),
no-store-with-ifs (*no-store*,
c-flag,
v-flag,
z-flag,
n-flag,
i-reg,

mar),
data-out-with-ifs (*data-out*,
a-reg,
b-reg,
c-flag,
i-reg,
mar),
reg-file (*reg-file*,
data-out,
i-reg,
mar,
no-store,
reset),
addr-out (*addr-out*, *reg-file*, *i-reg*, *mar*, *reset*),
c-flag-with-ifs (*c-flag*, *a-reg*, *b-reg*, *i-reg*, *mar*),
v-flag-with-ifs (*v-flag*,
a-reg,
b-reg,
c-flag,
i-reg,
mar),
z-flag-with-ifs (*z-flag*,
a-reg,
b-reg,
c-flag,
i-reg,
mar),
n-flag-with-ifs (*n-flag*,
a-reg,
b-reg,
c-flag,
i-reg,
mar),
a-reg (*a-reg*, *visual-mem*, *reg-file*, *i-reg*, *mar*, *reset*),
b-reg (*b-reg*, *visual-mem*, *reg-file*, *i-reg*, *mar*, *reset*),
i-reg-with-ifs (*i-reg*, *visual-mem*, *mar*),
visual-mem (*real-mem*,
read,
write,
addr-out,
memory-watch-dog-history,
dtack (car (*oracle*)),
reset (car (*oracle*))),
real-mem (*real-mem*,

read,
write,
addr-out,
data-out,
memory-watch-dog-history,
dtack (car (*oracle*)),
reset (car (*oracle*))),
watch-dog (*read*,
write,
dtack (car (*oracle*)),
data-out,
addr-out),
cdr (*oracle*)))

Event: Disable big-machine.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;;          This file contains definitions and lemmas concerning the          ;;
;;          sequencing of big-machine.  The proofs for the correctness of     ;;
;;          the micro-code are given here.                                    ;;
;;                                                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;;          Lemma used to tear apart "append"ed oracles                       ;;
;;          This is the fundamental lemma for sequencing big-machine.         ;;
;;          It permits us to express computations on a (append a b) in terms  ;;
;;          of a computation on b applied to the state produced by a.  We can ;;
;;          then get results about long computations by composing them from   ;;
;;          simple ones.                                                      ;;
;;                                                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

Theorem: big-machine-append
(m = big-machine (*mar*,
read,

write,
dtack,
reset,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
memory-watch-dog-history,
oracle1))

→ (big-machine (*mar*,
read,
write,
dtack,
reset,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
memory-watch-dog-history,
append (*oracle1*, *oracle2*))

= big-machine (car (*m*),
cadr (*m*),
caddr (*m*),
cadddr (*m*),
caddddr (*m*),
cadddddr (*m*),

caddddddr (m),
cadddddddr (m),
caddddddddr (m),
cadddddddddr (m),
caddddddddddr (m),
cadddddddddddr (m),
caddddddddddddr (m),
cadddddddddddddr (m),
caddddddddddddddr (m),
cadddddddddddddddr (m),
caddddddddddddddddr (m),
cadddddddddddddddddr (m),
caddddddddddddddddddr (m),
oracle2))

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;;          Miscellaneous lemmas about inc/dec and updating                   ;;
;;                                                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: bv-incrementer-truncates-to-machine-size
((*zero* = nat-to-bv (0, MACHINE-SIZE))
 ∧ bitvp (*bit-vec*)
 ∧ (size (*bit-vec*) = MACHINE-SIZE))
→ (bv-adder-output (**t**, *zero*, *bit-vec*) = v-nat-inc (*bit-vec*))

EVENT: Disable bv-incrementer-truncates-to-machine-size.

THEOREM: bv-decrementer-truncates-to-machine-size-help
((bv-to-nat (*bit-vec*) $\not<$ 1)
 ∧ bitvp (*bit-vec*)
 ∧ (size (*bit-vec*) = 16)
 ∧ (bv-to-nat (*bit-vec*) $\neq$ 0))
→ (nat-to-bv (bv-to-nat (*bit-vec*) − 1, 16)
    = nat-to-bv (bv-to-nat (*bit-vec*) − 1, 16))

THEOREM: bv-decrementer-truncates-to-machine-size
((*zero* = nat-to-bv (0, MACHINE-SIZE))
 ∧ bitvp (*bit-vec*)
 ∧ (size (*bit-vec*) = MACHINE-SIZE))
→ (bv-subtracter-output (**t**, *zero*, *bit-vec*) = v-nat-dec (*bit-vec*))

EVENT: Disable bv-decrementer-truncates-to-machine-size-help.

EVENT: Disable bv-decrementer-truncates-to-machine-size.

THEOREM: trunc-size-1
(n = size (a))
→ (trunc (a, n)
= **if** bitvp (a) **then** a
else BTM **endif**)

THEOREM: size-v-nth
((size (a) = 3)
∧ (length (*reg-file*) = 8)
∧ every-member-sizep (*reg-file*, MACHINE-SIZE))
→ (size (v-nth (a, *reg-file*)) = MACHINE-SIZE)

THEOREM: update-update-nth
update-nth (c, n, update-nth (c, n, *lst*, *value2*), *value1*)
= update-nth (c, n, *lst*, *value1*)

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;;          Resetting "big-machine"                                            ;;
;;                                                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: mar-on-reset
mar (*mar*, *i-reg*, *dtack*, **t**, *no-store*) = nat-to-bv (0, NXSZ)

THEOREM: reg-file-on-reset
reg-file (*reg-file*, *data-out*, *i-reg*, *mar*, *no-store*, **t**)
= update-nth (**t**, 0, *reg-file*, nat-to-bv (0, MACHINE-SIZE))

THEOREM: listp-reg-file
listp (reg-file (*reg-file*, *data-out*, *i-reg*, *mar*, *no-store*, *reset*))
= listp (*reg-file*)

```
;  The following lemma describes the first three clock "ticks" after a
;  reset occurs.  The memory-address register (MAR) is set to zero, the
;  read and write outputs are set to "false", and the reset latch is set
;  to "true".  Since nothing is know about the WATCH-DOG-HISTORY, it is
;  not possible to characterize what the memory state is.  However, after
```

```
;  these three clock "ticks" everything is stable (unchanging) until reset
;  is turned off.
```

THEOREM: reset-to-state-0
(standard-hyps (*mar*,
read,
write,
dtack,
reset,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem)
∧ (*m* = big-machine (*mar*,
read,
write,
dtack,
reset,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
memory-watch-dog-history,
list (list (*dt1*, **t**), list (*dt2*, **t**), list (*dt3*, **t**)))))
→ ((car (*m*) = nat-to-bv (0, NXSZ))

$\wedge$ (cadr (*m*) = **f**)
$\wedge$ (caddr (*m*) = **f**)
$\wedge$ (caddddr (*m*) = **t**)
$\wedge$ (nth (0, cadddddddr (*m*)) = nat-to-bv (0, MACHINE-SIZE)))

EVENT: Disable reset-to-state-0.

DEFINITION:
list-of-n-plus-1-dtack-off-reset-on (*count*)
= **if** *count* $\simeq$ 0 **then** list (list (**f**, **t**))
else cons (list (**f**, **t**),
list-of-n-plus-1-dtack-off-reset-on (*count* $-$ 1)) **endif**

THEOREM: state-0-to-0-wait-help
(*mar0* = nat-to-bv (0, NXSZ))
$\rightarrow$ (big-machine (*mar0*,
f,
f,
dtack,
t,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog-history,
cons (list (*dt-any*, **t**), *oracle*))
= big-machine (*mar0*,
f,
f,
fix-bool (*dt-any*),
t,
no-store,
data-out,
update-nth (**t**,
0,

reg-file,
nat-to-bv (0, MACHINE-SIZE)),
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, MACHINE-SIZE),
real-mem,
watch-dog (**f**, **f**, fix-bool (*dt-any*), *data-out*, *addr-out*),
oracle))

EVENT: Disable state-0-to-0-wait-help.

DEFINITION:
state-0-to-0-induction (*dtack*,
data-out,
reg-file,
addr-out,
visual-mem,
watch-dog,
n)
= **if** $n \simeq 0$ **then t**
else state-0-to-0-induction (**f**,
data-out,
update-nth (**t**,
0,
reg-file,
nat-to-bv (0,
MACHINE-SIZE)),
addr-out,
trunc (DEFAULT-VISUAL-MEM-VALUE,
MACHINE-SIZE),
watch-dog (**f**, **f**, **f**, *data-out*, *addr-out*),
$n - 1$) **endif**

THEOREM: state-0-to-0-wait
(*mar0* = nat-to-bv (0, NXSZ))
→ (big-machine (*mar0*,
f,
f,

dtack,
t,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog-history,
list-of-n-plus-1-dtack-off-reset-on (*n*))
= list (nat-to-bv (0, NXSZ),
f,
f,
f,
t,
no-store,
data-out,
update-nth (**t**, 0, *reg-file*, nat-to-bv (0, MACHINE-SIZE)),
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, MACHINE-SIZE),
real-mem,
watch-dog (**f**, **f**, **f**, *data-out*, *addr-out*)))

EVENT: Disable state-0-to-0-wait.

THEOREM: state-0-to-1
(*mar0* = nat-to-bv (0, NXSZ))
→ (big-machine (*mar0*,
f,
f,

dtack,
t,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
memory-watch-dog-history,
list (list (*dt-any1*, **f**), list (*dt-any2*, **f**)))

= list (nat-to-bv (1, NXSZ),
f,
f,
fix-bool (*dt-any2*),
f,
no-store,
data-out,
update-nth (**t**, 0, *reg-file*, nat-to-bv (0, MACHINE-SIZE)),
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, 16),
real-mem,
watch-dog (**f**, **f**, fix-bool (*dt-any2*), *data-out*, *addr-out*)))

EVENT: Disable state-0-to-1.

DEFINITION:
state-0-to-0-wait-to-1-oracle (*n*)
= append (list-of-n-plus-1-dtack-off-reset-on (*n*),
list (list (**f**, **f**), list (**f**, **f**)))

```
;  After the MAR has been set to zero, this lemma describes the steps taken by
;  FM8501 to prepare itself for executing instructions.
```

THEOREM: state-0-to-0-wait-to-1
(standard-hyps (*mar0*,
f,
f,
dtack,
t,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem)
∧ (*mar0* = nat-to-bv (0, NXSZ)))
→ (big-machine (*mar0*,
f,
f,
dtack,
t,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog-history,
state-0-to-0-wait-to-1-oracle (*n*))

= list (nat-to-bv (1, NXSZ),
f,
f,
f,
f,
no-store,
data-out,
update-nth (**t**, 0, *reg-file*, nat-to-bv (0, MACHINE-SIZE)),
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, 16),
real-mem,
watch-dog (**f**, **f**, **f**, *data-out*, *addr-out*)))

EVENT: Disable state-0-to-0-wait-to-1.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                              ;;
;;          "big-machine" instruction loop -- micro-state by micro-state        ;;
;;                                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                              ;;
;;          Fetch the instruction, and increment the PC                         ;;
;;                                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; We are about to embark on a litany of lemmas that move us from one microcode
; state to another.  By inspection of the oracles used below one can deduce the
; timing diagrams.  For example, the lemma below moves us from microcode state
; 1, where mar=1, and read, write, and reset are f, to microcode state 2, where
; mar=2, read, write, dtack and reset are f, the pc has been incremented,
; and addr-out contains the old pc.  To move from state 1 to 2, we need
; one clock tick with both dtack and reset f.

; We will similarly move through every state, some of which have several
; substates as we set lines, wait for acknowledgment, and then move.
```

```
; Once we get all the way through the microcode, we will append together
; the oracles to get an oracle that moves us all the way around from state 1
; to state 1.

; These lemmas make it perfectly clear what each state in the microcode does
; and how long each takes.
```

THEOREM: state-1-to-2
(ramp (*reg-file*, MACHINE-SIZE, 8) ∧ (*mar1* = nat-to-bv (1, NXSZ)))
→ (big-machine (*mar1*,
f,
f,
dtack,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
memory-watch-dog-history,
list (list (**f**, **f**)))
= list (nat-to-bv (2, NXSZ),
f,
f,
f,
f,
no-store,
data-out,
update-nth (**t**, 0, *reg-file*, v-nat-inc (nth (0, *reg-file*))),
nth (0, *reg-file*),
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,

b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, 16),
real-mem,
watch-dog (**f**, **f**, **f**, *data-out*, *addr-out*)))

EVENT: Disable state-1-to-2.

THEOREM: state-2-to-3-init
(*mar2* = nat-to-bv (2, NXSZ))
→ (big-machine (*mar2*,
f,
f,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog-history,
list (list (**f**, **f**)))
= list (nat-to-bv (2, NXSZ),
t,
f,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,

b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, 16),
real-mem,
watch-dog (**f**, **f**, **f**, *data-out*, *addr-out*)))

DEFINITION:
list-of-n-plus-1-dtack-reset-off (*count*)
= **if** *count* ≃ 0 **then** list (list (**f**, **f**))
else cons (list (**f**, **f**), list-of-n-plus-1-dtack-reset-off (*count* − 1)) **endif**

THEOREM: state-2-to-2-wait-help
(*mar2* = nat-to-bv (2, NXSZ))
→ (big-machine (*mar2*,
t,
f,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog-history,
cons (list (**f**, **f**), *oracle*))
= big-machine (*mar2*,
t,
f,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,

z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, MACHINE-SIZE),
real-mem,
watch-dog (**t**, **f**, **f**, *data-out*, *addr-out*),
oracle))

DEFINITION:
state-2-to-2-induction (*dtack*,
data-out,
addr-out,
visual-mem,
watch-dog-history,
n)
= **if** $n \simeq 0$ **then t**
else state-2-to-2-induction (**f**,
data-out,
addr-out,
trunc (DEFAULT-VISUAL-MEM-VALUE,
MACHINE-SIZE),
watch-dog (**t**, **f**, **f**, *data-out*, *addr-out*),
$n - 1$) **endif**

THEOREM: state-2-to-3-wait
(*mar2* = nat-to-bv (2, NXSZ))
→ (big-machine (*mar2*,
t,
f,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,

visual-mem,
real-mem,
watch-dog-history,
list-of-n-plus-1-dtack-reset-off (*n*))
= list (nat-to-bv (2, NXSZ),
t,
f,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, 16),
real-mem,
watch-dog (**t**, **f**, **f**, *data-out*, *addr-out*)))

EVENT: Disable state-2-to-3-wait.

THEOREM: upper-bound-bridge
(*n* = exp (2, size (*a*))) → (bv-to-nat (*a*) < *n*)

THEOREM: state-2-to-3-step3
(ramp (*real-mem*, MACHINE-SIZE, exp (2, MACHINE-SIZE))
∧ sizep (*addr-out*, MACHINE-SIZE)
∧ (*mar2* = nat-to-bv (2, NXSZ))
∧ (*visual-mem* = v-nth (*addr-out*, *real-mem*)))
→ (big-machine (*mar2*,
t,
f,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,

v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog (**t**, **f**, **f**, *data-out*, *addr-out*),
list (list (**t**, **f**), list (**t**, **f**)))

= list (nat-to-bv (3, NXSZ),
t,
f,
t,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
v-nth (*addr-out*, *real-mem*),
real-mem,
watch-dog (**t**, **f**, **t**, *data-out*, *addr-out*)))

EVENT: Disable state-2-to-3-step3.

THEOREM: state-3-to-4
(ramp (*real-mem*, MACHINE-SIZE, exp (2, MACHINE-SIZE))
∧ sizep (*addr-out*, MACHINE-SIZE)
∧ (*mar3* = nat-to-bv (3, NXSZ))
∧ (*visual-mem* = v-nth (*addr-out*, *real-mem*)))
→ (big-machine (*mar3*,
t,
f,
t,
f,
no-store,
data-out,

reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog (**t**, **f**, **t**, *data-out*, *addr-out*),
list (list (**t**, **f**)))

= list (nat-to-bv (4, NXSZ),
f,
f,
t,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
v-nth (*addr-out*, *real-mem*),
v-nth (*addr-out*, *real-mem*),
real-mem,
watch-dog (**t**, **f**, **t**, *data-out*, *addr-out*)))

EVENT: Disable state-3-to-4.

```
;  This proof is required because it is impossible to keep "APPEND"
;  from opening up on constants.  The opening does not allow
;  "BIG-MACHINE-APPEND" to apply because the 'append' will have
;  disappeared before it get a chance to apply.
```

THEOREM: state-2-to-3-step3-to-4
(ramp (*real-mem*, MACHINE-SIZE, exp (2, MACHINE-SIZE))
∧ sizep (*addr-out*, MACHINE-SIZE)

$$
\begin{array}{lll}
\wedge & (\mathit{mar2} = \text{nat-to-bv}\,(2,\ \textsc{nxsz}))) & \\
\rightarrow & (\text{big-machine}\,(\mathit{mar2}, & \\
 & & \mathbf{t},\\
 & & \mathbf{f},\\
 & & \mathbf{f},\\
 & & \mathbf{f},\\
 & & \mathit{no\text{-}store},\\
 & & \mathit{data\text{-}out},\\
 & & \mathit{reg\text{-}file},\\
 & & \mathit{addr\text{-}out},\\
 & & \mathit{c\text{-}flag},\\
 & & \mathit{v\text{-}flag},\\
 & & \mathit{z\text{-}flag},\\
 & & \mathit{n\text{-}flag},\\
 & & \mathit{a\text{-}reg},\\
 & & \mathit{b\text{-}reg},\\
 & & \mathit{i\text{-}reg},\\
 & & \mathit{visual\text{-}mem},\\
 & & \mathit{real\text{-}mem},\\
 & & \text{watch-dog}\,(\mathbf{t},\ \mathbf{f},\ \mathbf{f},\ \mathit{data\text{-}out},\ \mathit{addr\text{-}out}),\\
 & & \text{list}\,(\text{list}\,(\mathbf{t},\ \mathbf{f}),\ \text{list}\,(\mathbf{t},\ \mathbf{f}),\ \text{list}\,(\mathbf{t},\ \mathbf{f})))\\
 & = \quad \text{list}\,(\text{nat-to-bv}\,(4,\ \textsc{nxsz}), & \\
 & & \mathbf{f},\\
 & & \mathbf{f},\\
 & & \mathbf{t},\\
 & & \mathbf{f},\\
 & & \mathit{no\text{-}store},\\
 & & \mathit{data\text{-}out},\\
 & & \mathit{reg\text{-}file},\\
 & & \mathit{addr\text{-}out},\\
 & & \mathit{c\text{-}flag},\\
 & & \mathit{v\text{-}flag},\\
 & & \mathit{z\text{-}flag},\\
 & & \mathit{n\text{-}flag},\\
 & & \mathit{a\text{-}reg},\\
 & & \mathit{b\text{-}reg},\\
 & & \text{v-nth}\,(\mathit{addr\text{-}out},\ \mathit{real\text{-}mem}),\\
 & & \text{v-nth}\,(\mathit{addr\text{-}out},\ \mathit{real\text{-}mem}),\\
 & & \mathit{real\text{-}mem},\\
 & & \text{watch-dog}\,(\mathbf{t},\ \mathbf{f},\ \mathbf{t},\ \mathit{data\text{-}out},\ \mathit{addr\text{-}out})))
\end{array}
$$

EVENT: Disable state-2-to-3-step3-to-4.

DEFINITION:

```
fetch-ir-oracle (n)
=   append (list (list (f, f)),
            append (list (list (f, f)),
                    append (list-of-n-plus-1-dtack-reset-off (n),
                            list (list (t, f), list (t, f), list (t, f)))))

THEOREM: state-1-to-4
 (ramp (reg-file, MACHINE-SIZE, 8)
  ∧  sizep (addr-out, MACHINE-SIZE)
  ∧  ramp (real-mem, MACHINE-SIZE, exp (2, MACHINE-SIZE))
  ∧  (mar1 = nat-to-bv (1, NXSZ)))
 →  (big-machine (mar1,
                  f,
                  f,
                  f,
                  f,
                  no-store,
                  data-out,
                  reg-file,
                  addr-out,
                  c-flag,
                  v-flag,
                  z-flag,
                  n-flag,
                  a-reg,
                  b-reg,
                  i-reg,
                  visual-mem,
                  real-mem,
                  memory-watch-dog-history,
                  fetch-ir-oracle (n))
     =  list (nat-to-bv (4, NXSZ),
              f,
              f,
              t,
              f,
              no-store,
              data-out,
              update-nth (t, 0, reg-file, v-nat-inc (nth (0, reg-file))),
              nth (0, reg-file),
              c-flag,
              v-flag,
              z-flag,
              n-flag,
```

a-reg,
b-reg,
v-nth (nth (0, *reg-file*), *real-mem*),
v-nth (nth (0, *reg-file*), *real-mem*),
real-mem,
watch-dog (**t**, **f**, **t**, *data-out*, nth (0, *reg-file*))))

EVENT: Disable state-1-to-4.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                              ;;
;;          Some size lemmas                                                    ;;
;;                                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: size-bv-reg-select
size (bv-reg-select (*i-reg*, *mar*, *reset*)) = 3

THEOREM: size-bv-oprd-a
size (bv-oprd-a (*i-reg*)) = 3

THEOREM: size-bv-oprd-b
size (bv-oprd-b (*i-reg*)) = 3

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                              ;;
;;          Fetch operand A, doing pre-decrement if necessary                   ;;
;;                                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:
fetch-oprd-a-reg-file (*i-reg*, *reg-file*)
= update-v-nth (b-indirect-reg-a-dec (*i-reg*),
bv-oprd-a (*i-reg*),
reg-file,
v-nat-dec (v-nth (bv-oprd-a (*i-reg*), *reg-file*)))

THEOREM: every-member-sizep-update-nth
((size (*x*) = *width*) ∧ every-member-sizep (*lst*, *width*))
→ every-member-sizep (update-nth (*bool*, *n*, *lst*, *x*), *width*)

```
;  Both of the following two lemmas are needed, as we sometimes have RAMP enabled.
```

THEOREM: ramp-fetch-oprd-a-reg-file
((*m* = MACHINE-SIZE) ∧ ramp (*reg-file*, MACHINE-SIZE, 8))
→ ((length (fetch-oprd-a-reg-file (*i-reg*, *reg-file*)) = 8)
∧ every-member-sizep (fetch-oprd-a-reg-file (*i-reg*, *reg-file*), *m*))

THEOREM: ramp-fetch-oprd-a-reg-file-2
((*m* = MACHINE-SIZE) ∧ ramp (*reg-file*, MACHINE-SIZE, 8))
→ ramp (fetch-oprd-a-reg-file (*i-reg*, *reg-file*), *m*, 8)

DEFINITION:
fetch-oprd-a-addr-out (*i-reg*, *reg-file*)
= **if** b-indirect-reg-a-dec (*i-reg*)
then v-nat-dec (v-nth (bv-oprd-a (*i-reg*), *reg-file*))
else v-nth (bv-oprd-a (*i-reg*), *reg-file*) **endif**

THEOREM: sizep-fetch-oprd-a-addr-out
((*m* = MACHINE-SIZE) ∧ ramp (*reg-file*, MACHINE-SIZE, 8))
→ (size (fetch-oprd-a-addr-out (*i-reg*, *reg-file*)) = *m*)

THEOREM: state-4-to-5
(ramp (*reg-file*, MACHINE-SIZE, 8) ∧ (*mar4* = nat-to-bv (4, NXSZ)))
→ (big-machine (*mar4*,
f,
f,
t,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog,
list (list (**f**, **f**)))
= list (nat-to-bv (5, NXSZ),

f,
f,
f,
f,
no-store,
data-out,
fetch-oprd-a-reg-file (*i-reg*, *reg-file*),
fetch-oprd-a-addr-out (*i-reg*, *reg-file*),
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, 16),
real-mem,
watch-dog (**f**, **f**, **f**, *data-out*, *addr-out*)))

EVENT: Disable state-4-to-5.

```
;  The next two prove-lemmas are of type nil -- we never use them as
;  rewrite rules.  Ideally we would like to append together the two clocks
;  for state-5-to-6-reg and state-6-to-7-reg.  But the two clocks are both
;  constant so the append computes and big-machine-append will not apply.
;  So we will actually prove state-4-to-5-to-6-to-7-reg as one rewrite rule.
;  But for the sake of exposition, we prove individual lemmas about each
;  state transition.
```

THEOREM: state-5-to-6-reg
(ramp (*reg-file*, MACHINE-SIZE, 8)
 ∧ (*mar5* = nat-to-bv (5, NXSZ))
 ∧ b-direct-reg-a (*i-reg*))
→ (big-machine (*mar5*,
f,
f,
dt-any,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,

v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog-history,
list (list (**f**, **f**)))
= list (nat-to-bv (6, NXSZ),
f,
f,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, 16),
real-mem,
watch-dog (**f**, **f**, **f**, *data-out*, *addr-out*)))

THEOREM: state-6-to-7-reg
(ramp (*reg-file*, MACHINE-SIZE, 8)
∧ (*mar6* = nat-to-bv (6, NXSZ))
∧ b-direct-reg-a (*i-reg*))
→ (big-machine (*mar6*,
f,
f,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,

v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog-history,
list (list (**f**, **f**)))
= list (nat-to-bv (7, NXSZ),
f,
f,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
v-nth (bv-oprd-a (*i-reg*), *reg-file*),
b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, 16),
real-mem,
watch-dog (**f**, **f**, **f**, *data-out*, *addr-out*)))

THEOREM: state-5-to-6-mem-init
((*mar5* = nat-to-bv (5, NXSZ)) ∧ (¬ b-direct-reg-a (*i-reg*)))
→ (big-machine (*mar5*,
f,
f,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,

n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
memory-watch-dog-history,
list (list (**f**, **f**)))
= list (nat-to-bv (5, NXSZ),
t,
f,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, 16),
real-mem,
watch-dog (**f**, **f**, **f**, *data-out*, *addr-out*)))

EVENT: Disable state-5-to-6-mem-init.

THEOREM: state-5-to-6-mem-help
((*mar5* = nat-to-bv (5, NXSZ)) ∧ (¬ b-direct-reg-a (*i-reg*)))
→ (big-machine (*mar5*,
t,
f,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,

n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog-history,
cons (list (**f**, **f**), *oracle*))
= big-machine (*mar5*,
t,
f,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, MACHINE-SIZE),
real-mem,
watch-dog (**t**, **f**, **f**, *data-out*, *addr-out*),
oracle))

EVENT: Disable state-5-to-6-mem-help.

THEOREM: state-5-to-6-mem-wait
((*mar5* = nat-to-bv (5, NXSZ)) ∧ (¬ b-direct-reg-a (*i-reg*)))
→ (big-machine (*mar5*,
t,
f,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,

z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog-history,
list-of-n-plus-1-dtack-reset-off (*n*))
= list (nat-to-bv (5, NXSZ),
t,
f,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, 16),
real-mem,
watch-dog (**t**, **f**, **f**, *data-out*, *addr-out*)))

EVENT: Disable state-5-to-6-mem-wait.

```
; The next two lemmas we never use as rewrite rules, for the same reasons
; explained above.
```

THEOREM: state-5-to-6-mem-dtack
((*mar5* = nat-to-bv (5, NXSZ))
∧ ramp (*reg-file*, MACHINE-SIZE, 8)
∧ sizep (*addr-out*, MACHINE-SIZE)
∧ (¬ b-direct-reg-a (*i-reg*)))
→ (big-machine (*mar5*,
t,
f,
f,

f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog (**t**, **f**, **f**, *data-out*, *addr-out*),
list (list (**t**, **f**), list (**t**, **f**)))

$=$ list (nat-to-bv (6, NXSZ),
t,
f,
t,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
trunc (v-nth (*addr-out*, *real-mem*), MACHINE-SIZE),
real-mem,
watch-dog (**t**, **f**, **t**, *data-out*, *addr-out*)))

THEOREM: state-6-to-7-mem
((*mar6* = nat-to-bv (6, NXSZ))
$\wedge$ ramp (*reg-file*, MACHINE-SIZE, 8)
$\wedge$ sizep (*visual-mem*, MACHINE-SIZE)
$\wedge$ sizep (*addr-out*, MACHINE-SIZE)
$\wedge$ ($\neg$ b-direct-reg-a (*i-reg*)))
$\rightarrow$ (big-machine (*mar6*,
t,

f,
t,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog (**t**, **f**, **t**, *data-out*, *addr-out*),
list (list (**t**, **f**)))

= list (nat-to-bv (7, NXSZ),
f,
f,
t,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
visual-mem,
b-reg,
i-reg,
trunc (v-nth (*addr-out*, *real-mem*), MACHINE-SIZE),
real-mem,
watch-dog (**t**, **f**, **t**, *data-out*, *addr-out*)))

```
; We now piece together states 4, 5, 6, and 7 for the reg case:
```

THEOREM: state-4-to-5-to-6-to-7-reg
(ramp (*reg-file*, MACHINE-SIZE, 8)
∧ (*mar4* = nat-to-bv (4, NXSZ))

$\wedge$ b-direct-reg-a (*i-reg*))
$\rightarrow$ (big-machine (*mar4*,
f,
f,
t,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog,
list (list (**f**, **f**), list (**f**, **f**), list (**f**, **f**)))
= list (nat-to-bv (7, NXSZ),
f,
f,
f,
f,
no-store,
data-out,
fetch-oprd-a-reg-file (*i-reg*, *reg-file*),
fetch-oprd-a-addr-out (*i-reg*, *reg-file*),
c-flag,
v-flag,
z-flag,
n-flag,
v-nth (bv-oprd-a (*i-reg*),
fetch-oprd-a-reg-file (*i-reg*, *reg-file*)),
b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, 16),
real-mem,
watch-dog (**f**,
f,
f,
data-out,

fetch-oprd-a-addr-out (*i-reg*, *reg-file*))))

EVENT: Disable state-4-to-5-to-6-to-7-reg.

```
; and now we do the same thing for 5, 6, and 7 in the mem case:
```

THEOREM: state-5-to-6-to-7-mem-dtack
((*mar5* = nat-to-bv (5, NXSZ))
∧ ramp (*reg-file*, MACHINE-SIZE, 8)
∧ sizep (*addr-out*, MACHINE-SIZE)
∧ (¬ b-direct-reg-a (*i-reg*)))
→ (big-machine (*mar5*,
t,
f,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog (**t**, **f**, **f**, *data-out*, *addr-out*),
list (list (**t**, **f**), list (**t**, **f**), list (**t**, **f**)))
= list (nat-to-bv (7, NXSZ),
f,
f,
t,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,

n-flag,
trunc (v-nth (*addr-out*, *real-mem*), MACHINE-SIZE),
b-reg,
i-reg,
trunc (v-nth (*addr-out*, *real-mem*), MACHINE-SIZE),
real-mem,
watch-dog (**t**, **f**, **t**, *data-out*, *addr-out*)))

EVENT: Disable state-5-to-6-to-7-mem-dtack.

```
; Finally, we do the entire transition from state 4 to state 7.  We first
; define the oracle necessary to drive the machine through the sequence.
```

DEFINITION:
fetch-oprd-a-oracle (*i-reg*, *n*)
= **if** b-direct-reg-a (*i-reg*) **then** list (list (**f**, **f**), list (**f**, **f**), list (**f**, **f**))
else append (list (list (**f**, **f**)),
append (list (list (**f**, **f**)),
append (list-of-n-plus-1-dtack-reset-off (*n*),
list (list (**t**, **f**), list (**t**, **f**), list (**t**, **f**))))) **endif**

THEOREM: every-member-sizep-update-nth-machine-size
((size (*x*) = 16) ∧ every-member-sizep (*lst*, 16))
→ every-member-sizep (update-nth (*bool*, *n*, *lst*, *x*), 16)

EVENT: Disable every-member-sizep-update-nth.

THEOREM: state-4-to-7
(ramp (*reg-file*, MACHINE-SIZE, 8)
∧ sizep (*addr-out*, MACHINE-SIZE)
∧ ramp (*real-mem*, MACHINE-SIZE, exp (2, MACHINE-SIZE))
∧ (*mar4* = nat-to-bv (4, NXSZ)))
→ (big-machine (*mar4*,
f,
f,
t,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,

z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog,
fetch-oprd-a-oracle (*i-reg*, *n*))

= list (nat-to-bv (7, NXSZ),
f,
f,
¬ b-direct-reg-a (*i-reg*),
f,
no-store,
data-out,
fetch-oprd-a-reg-file (*i-reg*, *reg-file*),
fetch-oprd-a-addr-out (*i-reg*, *reg-file*),
c-flag,
v-flag,
z-flag,
n-flag,
if b-direct-reg-a (*i-reg*)
then v-nth (bv-oprd-a (*i-reg*),
fetch-oprd-a-reg-file (*i-reg*, *reg-file*))
else v-nth (fetch-oprd-a-addr-out (*i-reg*, *reg-file*),
real-mem) **endif**,
b-reg,
i-reg,
if b-direct-reg-a (*i-reg*)
then trunc (DEFAULT-VISUAL-MEM-VALUE, 16)
else v-nth (fetch-oprd-a-addr-out (*i-reg*, *reg-file*),
real-mem) **endif**,
real-mem,
watch-dog (¬ b-direct-reg-a (*i-reg*),
f,
¬ b-direct-reg-a (*i-reg*),
data-out,
fetch-oprd-a-addr-out (*i-reg*, *reg-file*))))

EVENT: Disable state-4-to-7.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                             ;;
```

```
;;          Fetch operand B, doing pre-decrement if necessary                  ;;
;;                                                                             ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:
fetch-oprd-b-reg-file (*i-reg*, *reg-file*)
= update-v-nth (b-indirect-reg-b-dec (*i-reg*),
bv-oprd-b (*i-reg*),
reg-file,
v-nat-dec (v-nth (bv-oprd-b (*i-reg*), *reg-file*)))

```
; Both of the following lemmas are needed, as RAMP is enabled sometimes
```

THEOREM: ramp-fetch-oprd-b-reg-file
((*m* = MACHINE-SIZE) ∧ ramp (*reg-file*, MACHINE-SIZE, 8))
→ ((length (fetch-oprd-b-reg-file (*i-reg*, *reg-file*)) = 8)
∧ every-member-sizep (fetch-oprd-b-reg-file (*i-reg*, *reg-file*), *m*))

THEOREM: ramp-fetch-oprd-b-reg-file-2
((*m* = MACHINE-SIZE) ∧ ramp (*reg-file*, MACHINE-SIZE, 8))
→ ramp (fetch-oprd-b-reg-file (*i-reg*, *reg-file*), *m*, 8)

DEFINITION:
fetch-oprd-b-addr-out (*i-reg*, *reg-file*)
= **if** b-indirect-reg-b-dec (*i-reg*)
then v-nat-dec (v-nth (bv-oprd-b (*i-reg*), *reg-file*))
else v-nth (bv-oprd-b (*i-reg*), *reg-file*) **endif**

THEOREM: sizep-fetch-oprd-b-addr-out
((*m* = MACHINE-SIZE) ∧ ramp (*reg-file*, MACHINE-SIZE, 8))
→ (size (fetch-oprd-b-addr-out (*i-reg*, *reg-file*)) = *m*)

THEOREM: state-7-to-8
(ramp (*reg-file*, MACHINE-SIZE, 8) ∧ (*mar7* = nat-to-bv (7, NXSZ)))
→ (big-machine (*mar7*,
f,
f,
dtack,
f,
no-store,
data-out,

reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog,
list (list (**f**, **f**)))
= list (nat-to-bv (8, NXSZ),
f,
f,
f,
f,
no-store,
data-out,
fetch-oprd-b-reg-file (*i-reg*, *reg-file*),
fetch-oprd-b-addr-out (*i-reg*, *reg-file*),
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, 16),
real-mem,
watch-dog (**f**, **f**, **f**, *data-out*, *addr-out*)))

EVENT: Disable state-7-to-8.

```
; The next two lemmas are never used as rewrite rules.
```

THEOREM: state-8-to-9-reg
(ramp (*reg-file*, MACHINE-SIZE, 8)
∧ (*mar8* = nat-to-bv (8, NXSZ))
∧ b-direct-reg-b (*i-reg*))
→ (big-machine (*mar8*,
f,

f,
dt-any,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog-history,
list (list (**f**, **f**)))
= list (nat-to-bv (9, NXSZ),
f,
f,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, 16),
real-mem,
watch-dog (**f**, **f**, **f**, *data-out*, *addr-out*)))

THEOREM: state-9-to-10-reg
(ramp (*reg-file*, MACHINE-SIZE, 8)
∧ (*mar9* = nat-to-bv (9, NXSZ))
∧ b-direct-reg-b (*i-reg*))
→ (big-machine (*mar9*,
f,

f,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog-history,
list (list (**f**, **f**)))
= list (nat-to-bv (10, NXSZ),
f,
f,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
v-nth (bv-oprd-b (*i-reg*), *reg-file*),
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, 16),
real-mem,
watch-dog (**f**, **f**, **f**, *data-out*, *addr-out*)))

THEOREM: state-8-to-9-mem-init
((*mar8* = nat-to-bv (8, NXSZ)) ∧ (¬ b-direct-reg-b (*i-reg*)))
→ (big-machine (*mar8*,
f,
f,
f,

f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
memory-watch-dog-history,
list (list (**f**, **f**)))
= list (nat-to-bv (8, NXSZ),
t,
f,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, 16),
real-mem,
watch-dog (**f**, **f**, **f**, *data-out*, *addr-out*)))

EVENT: Disable state-8-to-9-mem-init.

THEOREM: state-8-to-9-mem-help
((*mar8* = nat-to-bv (8, NXSZ)) ∧ (¬ b-direct-reg-b (*i-reg*)))
→ (big-machine (*mar8*,
t,
f,
f,

f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog-history,
cons (list (**f**, **f**), *oracle*))
= big-machine (*mar8*,
t,
f,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, MACHINE-SIZE),
real-mem,
watch-dog (**t**, **f**, **f**, *data-out*, *addr-out*),
oracle))

EVENT: Disable state-8-to-9-mem-help.

THEOREM: state-8-to-9-mem-wait
((*mar8* = nat-to-bv (8, NXSZ)) ∧ (¬ b-direct-reg-b (*i-reg*)))
→ (big-machine (*mar8*,
t,
f,

f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog-history,
list-of-n-plus-1-dtack-reset-off (*n*))
= list (nat-to-bv (8, NXSZ),
t,
f,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, 16),
real-mem,
watch-dog (**t**, **f**, **f**, *data-out*, *addr-out*)))

EVENT: Disable state-8-to-9-mem-wait.

```
; the next two are never used as rewrites.
```

THEOREM: state-8-to-9-mem-dtack
((*mar8* = nat-to-bv (8, NXSZ))

$\wedge$ ramp (*reg-file*, MACHINE-SIZE, 8)
$\wedge$ sizep (*addr-out*, MACHINE-SIZE)
$\wedge$ ($\neg$ b-direct-reg-b (*i-reg*)))
$\rightarrow$ (big-machine (*mar8*,
t,
f,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog (**t**, **f**, **f**, *data-out*, *addr-out*),
list (list (**t**, **f**), list (**t**, **f**)))
= list (nat-to-bv (9, NXSZ),
t,
f,
t,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
trunc (v-nth (*addr-out*, *real-mem*), MACHINE-SIZE),
real-mem,
watch-dog (**t**, **f**, **t**, *data-out*, *addr-out*)))

THEOREM: state-9-to-10-mem

$$
\begin{array}{lll}
((\mathit{mar9} = \text{nat-to-bv}\,(\texttt{9},\ \textsc{nxsz})) & & \\
\wedge & \text{ramp}\,(\mathit{reg\text{-}file},\ \textsc{machine-size},\ \texttt{8}) & \\
\wedge & \text{sizep}\,(\mathit{visual\text{-}mem},\ \textsc{machine-size}) & \\
\wedge & \text{sizep}\,(\mathit{addr\text{-}out},\ \textsc{machine-size}) & \\
\wedge & (\neg\ \text{b-direct-reg-b}\,(\mathit{i\text{-}reg}))) & \\
\rightarrow & (\text{big-machine}\,(\mathit{mar9}, & \\
 & \qquad\qquad \mathbf{t}, & \\
 & \qquad\qquad \mathbf{f}, & \\
 & \qquad\qquad \mathbf{t}, & \\
 & \qquad\qquad \mathbf{f}, & \\
 & \qquad\qquad \mathit{no\text{-}store}, & \\
 & \qquad\qquad \mathit{data\text{-}out}, & \\
 & \qquad\qquad \mathit{reg\text{-}file}, & \\
 & \qquad\qquad \mathit{addr\text{-}out}, & \\
 & \qquad\qquad \mathit{c\text{-}flag}, & \\
 & \qquad\qquad \mathit{v\text{-}flag}, & \\
 & \qquad\qquad \mathit{z\text{-}flag}, & \\
 & \qquad\qquad \mathit{n\text{-}flag}, & \\
 & \qquad\qquad \mathit{a\text{-}reg}, & \\
 & \qquad\qquad \mathit{b\text{-}reg}, & \\
 & \qquad\qquad \mathit{i\text{-}reg}, & \\
 & \qquad\qquad \mathit{visual\text{-}mem}, & \\
 & \qquad\qquad \mathit{real\text{-}mem}, & \\
 & \qquad\qquad \text{watch-dog}\,(\mathbf{t},\ \mathbf{f},\ \mathbf{t},\ \mathit{data\text{-}out},\ \mathit{addr\text{-}out}), & \\
 & \qquad\qquad \text{list}\,(\text{list}\,(\mathbf{t},\ \mathbf{f}))) & \\
 & = \quad \text{list}\,(\text{nat-to-bv}\,(\texttt{10},\ \textsc{nxsz}), & \\
 & \qquad\quad \mathbf{f}, & \\
 & \qquad\quad \mathbf{f}, & \\
 & \qquad\quad \mathbf{t}, & \\
 & \qquad\quad \mathbf{f}, & \\
 & \qquad\quad \mathit{no\text{-}store}, & \\
 & \qquad\quad \mathit{data\text{-}out}, & \\
 & \qquad\quad \mathit{reg\text{-}file}, & \\
 & \qquad\quad \mathit{addr\text{-}out}, & \\
 & \qquad\quad \mathit{c\text{-}flag}, & \\
 & \qquad\quad \mathit{v\text{-}flag}, & \\
 & \qquad\quad \mathit{z\text{-}flag}, & \\
 & \qquad\quad \mathit{n\text{-}flag}, & \\
 & \qquad\quad \mathit{a\text{-}reg}, & \\
 & \qquad\quad \mathit{visual\text{-}mem}, & \\
 & \qquad\quad \mathit{i\text{-}reg}, & \\
 & \qquad\quad \text{trunc}\,(\text{v-nth}\,(\mathit{addr\text{-}out},\ \mathit{real\text{-}mem}),\ \textsc{machine-size}), & \\
 & \qquad\quad \mathit{real\text{-}mem}, & \\
 & \qquad\quad \text{watch-dog}\,(\mathbf{t},\ \mathbf{f},\ \mathbf{t},\ \mathit{data\text{-}out},\ \mathit{addr\text{-}out}))) &
\end{array}
$$

```
; We now piece together states 7, 8, 9, and 10 for the reg case:
```

THEOREM: state-7-to-8-to-9-to-10-reg
(ramp (*reg-file*, MACHINE-SIZE, 8)
 ∧ (*mar7* = nat-to-bv (7, NXSZ))
 ∧ b-direct-reg-b (*i-reg*))
→ (big-machine (*mar7*,
                f,
                f,
                dtack,
                f,
                no-store,
                data-out,
                reg-file,
                addr-out,
                c-flag,
                v-flag,
                z-flag,
                n-flag,
                a-reg,
                b-reg,
                i-reg,
                visual-mem,
                real-mem,
                watch-dog,
                list (list (**f**, **f**), list (**f**, **f**), list (**f**, **f**)))
   = list (nat-to-bv (10, NXSZ),
           f,
           f,
           f,
           f,
           no-store,
           data-out,
           fetch-oprd-b-reg-file (*i-reg*, *reg-file*),
           fetch-oprd-b-addr-out (*i-reg*, *reg-file*),
           c-flag,
           v-flag,
           z-flag,
           n-flag,
           a-reg,
           v-nth (bv-oprd-b (*i-reg*),
                  fetch-oprd-b-reg-file (*i-reg*, *reg-file*)),
           i-reg,

trunc (DEFAULT-VISUAL-MEM-VALUE, 16),
real-mem,
watch-dog (**f**,
f,
f,
data-out,
fetch-oprd-b-addr-out (*i-reg*, *reg-file*))))

EVENT: Disable state-7-to-8-to-9-to-10-reg.

```
; and now we do the same thing for 8, 9 and 10 in the mem case.
```

THEOREM: state-8-to-9-to-10-mem-dtack
((*mar8* = nat-to-bv (8, NXSZ))
∧ ramp (*reg-file*, MACHINE-SIZE, 8)
∧ sizep (*addr-out*, MACHINE-SIZE)
∧ (¬ b-direct-reg-b (*i-reg*)))
→ (big-machine (*mar8*,
t,
f,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog (**t**, **f**, **f**, *data-out*, *addr-out*),
list (list (**t**, **f**), list (**t**, **f**), list (**t**, **f**)))
= list (nat-to-bv (10, NXSZ),
f,
f,
t,
f,
no-store,

data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
trunc (v-nth (*addr-out*, *real-mem*), MACHINE-SIZE),
i-reg,
trunc (v-nth (*addr-out*, *real-mem*), MACHINE-SIZE),
real-mem,
watch-dog (**t**, **f**, **t**, *data-out*, *addr-out*)))

EVENT: Disable state-5-to-6-to-7-mem-dtack.

```
; Finally, we do the entire transition from state 7 to state 10.  We first
; define the oracle necessary to drive the machine through the sequence.
```

DEFINITION:
fetch-oprd-b-oracle (*i-reg*, *n*)
= **if** b-direct-reg-b (*i-reg*) **then** list (list (**f**, **f**), list (**f**, **f**), list (**f**, **f**))
else append (list (list (**f**, **f**)),
append (list (list (**f**, **f**)),
append (list-of-n-plus-1-dtack-reset-off (*n*),
list (list (**t**, **f**), list (**t**, **f**), list (**t**, **f**))))) **endif**

THEOREM: state-7-to-10
(ramp (*reg-file*, MACHINE-SIZE, 8)
∧ sizep (*addr-out*, MACHINE-SIZE)
∧ ramp (*real-mem*, MACHINE-SIZE, exp (2, MACHINE-SIZE))
∧ (*mar7* = nat-to-bv (7, NXSZ)))
→ (big-machine (*mar7*,
f,
f,
dtack,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,

```
                z-flag,
                n-flag,
                a-reg,
                b-reg,
                i-reg,
                visual-mem,
                real-mem,
                watch-dog,
                fetch-oprd-b-oracle (i-reg, n))
=   list (nat-to-bv (10, NXSZ),
          f,
          f,
          ¬ b-direct-reg-b (i-reg),
          f,
          no-store,
          data-out,
          fetch-oprd-b-reg-file (i-reg, reg-file),
          fetch-oprd-b-addr-out (i-reg, reg-file),
          c-flag,
          v-flag,
          z-flag,
          n-flag,
          a-reg,
          if b-direct-reg-b (i-reg)
          then v-nth (bv-oprd-b (i-reg),
                      fetch-oprd-b-reg-file (i-reg, reg-file))
          else v-nth (fetch-oprd-b-addr-out (i-reg, reg-file),
                     real-mem) endif,
          i-reg,
          if b-direct-reg-b (i-reg)
          then trunc (DEFAULT-VISUAL-MEM-VALUE, 16)
          else v-nth (fetch-oprd-b-addr-out (i-reg, reg-file),
                     real-mem) endif,
          real-mem,
          watch-dog (¬ b-direct-reg-b (i-reg),
                     f,
                     ¬ b-direct-reg-b (i-reg),
                     data-out,
                     fetch-oprd-b-addr-out (i-reg, reg-file))))
```

EVENT: Disable state-7-to-10.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
```

```
;;                    ALU, results to memory or register                        ;;
;;                                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:
alu-c-flag (*i-reg*, *a-reg*, *b-reg*, *c-flag*)
= **if** b-cc-set (*i-reg*)
  then c (bv-alu-cv (*a-reg*, *b-reg*, *c-flag*, bv-alu-op-code (*i-reg*)))
  else *c-flag* **endif**

DEFINITION:
alu-v-flag (*i-reg*, *a-reg*, *b-reg*, *c-flag*, *v-flag*)
= **if** b-cc-set (*i-reg*)
  then v (bv-alu-cv (*a-reg*, *b-reg*, *c-flag*, bv-alu-op-code (*i-reg*)))
  else *v-flag* **endif**

DEFINITION:
alu-z-flag (*i-reg*, *a-reg*, *b-reg*, *c-flag*, *z-flag*)
= **if** b-cc-set (*i-reg*)
  then b-bv-zerop (bv (bv-alu-cv (*a-reg*, *b-reg*, *c-flag*, bv-alu-op-code (*i-reg*))))
  else *z-flag* **endif**

DEFINITION:
alu-n-flag (*i-reg*, *a-reg*, *b-reg*, *c-flag*, *n-flag*)
= **if** b-cc-set (*i-reg*)
  then bitn (bv (bv-alu-cv (*a-reg*, *b-reg*, *c-flag*, bv-alu-op-code (*i-reg*))),
          MACHINE-SIZE)
  else *n-flag* **endif**

THEOREM: boolp-alu-c-flag
boolp (*c-flag*) → boolp (alu-c-flag (*i-reg*, *a-reg*, *b-reg*, *c-flag*))

THEOREM: boolp-alu-v-flag
boolp (*v-flag*) → boolp (alu-v-flag (*i-reg*, *a-reg*, *b-reg*, *c-flag*, *v-flag*))

THEOREM: boolp-alu-z-flag
boolp (*z-flag*) → boolp (alu-z-flag (*i-reg*, *a-reg*, *b-reg*, *c-flag*, *z-flag*))

THEOREM: boolp-alu-n-flag
boolp (*n-flag*) → boolp (alu-n-flag (*i-reg*, *a-reg*, *b-reg*, *c-flag*, *n-flag*))

THEOREM: state-10-to-11
((*mar10* = nat-to-bv (10, NXSZ))
 ∧ boolp (*no-store*)

$\wedge$ boolp (*c-flag*)
$\wedge$ boolp (*v-flag*)
$\wedge$ boolp (*n-flag*)
$\wedge$ boolp (*z-flag*)
$\wedge$ sizep (*data-out*, MACHINE-SIZE))
$\rightarrow$ (big-machine (*mar10*,
f,
f,
dtack,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog-history,
list (list (**f**, **f**)))
= list (nat-to-bv (11, NXSZ),
f,
f,
f,
f,
$\neg$ b-store-alu-result (*c-flag*, *v-flag*, *z-flag*, *n-flag*, *i-reg*),
bv (bv-alu-cv (*a-reg*, *b-reg*, *c-flag*, bv-alu-op-code (*i-reg*))),
reg-file,
addr-out,
alu-c-flag (*i-reg*, *a-reg*, *b-reg*, *c-flag*),
alu-v-flag (*i-reg*, *a-reg*, *b-reg*, *c-flag*, *v-flag*),
alu-z-flag (*i-reg*, *a-reg*, *b-reg*, *c-flag*, *z-flag*),
alu-n-flag (*i-reg*, *a-reg*, *b-reg*, *c-flag*, *n-flag*),
a-reg,
b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, 16),
real-mem,
watch-dog (**f**, **f**, **f**, *data-out*, *addr-out*)))

EVENT: Disable state-10-to-11.

EVENT: Disable alu-c-flag.

EVENT: Disable alu-v-flag.

EVENT: Disable alu-z-flag.

EVENT: Disable alu-n-flag.

THEOREM: state-11-to-12-reg
(ramp (*reg-file*, MACHINE-SIZE, 8)
∧ sizep (*data-out*, MACHINE-SIZE)
∧ (*mar11* = nat-to-bv (11, NXSZ))
∧ b-direct-reg-b (*i-reg*))
→ (big-machine (*mar11*,
f,
f,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog-history,
list (list (**f**, **f**)))
= list (nat-to-bv (12, NXSZ),
f,
f,
f,
f,
no-store,

data-out,
update-v-nth (¬ *no-store*,
bv-oprd-b (*i-reg*),
reg-file,
data-out),
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, 16),
real-mem,
watch-dog (**f**, **f**, **f**, *data-out*, *addr-out*)))

THEOREM: state-11-to-12-mem-no-store
((*mar11* = nat-to-bv (11, NXSZ))
∧ (*no-store* = **t**)
∧ (¬ b-direct-reg-b (*i-reg*)))
→ (big-machine (*mar11*,
f,
f,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog-history,
list (list (**f**, **f**)))
= list (nat-to-bv (12, NXSZ),
f,
f,

f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, 16),
real-mem,
watch-dog (**f**, **f**, **f**, *data-out*, *addr-out*)))

THEOREM: state-11-to-12-mem-init-store
((*mar11* = nat-to-bv (11, NXSZ))
∧ (*no-store* = **f**)
∧ (¬ b-direct-reg-b (*i-reg*)))
→ (big-machine (*mar11*,
f,
f,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog-history,
list (list (**f**, **f**)))
= list (nat-to-bv (11, NXSZ),
f,
t,

f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, 16),
real-mem,
watch-dog (**f**, **f**, **f**, *data-out*, *addr-out*)))

EVENT: Disable state-11-to-12-mem-init-store.

THEOREM: state-11-to-12-mem-help
((*mar11* = nat-to-bv (11, NXSZ))
∧ (*no-store* = **f**)
∧ (¬ b-direct-reg-b (*i-reg*)))
→ (big-machine (*mar11*,
f,
t,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog-history,
cons (list (**f**, **f**), *oracle*))
= big-machine (*mar11*,

f,
t,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, MACHINE-SIZE),
real-mem,
watch-dog (**f**, **t**, **f**, *data-out*, *addr-out*),
oracle))

EVENT: Disable state-11-to-12-mem-help.

DEFINITION:
state-11-to-11-induction (*dtack*,
data-out,
addr-out,
visual-mem,
watch-dog-history,
n)
= **if** $n \simeq 0$ **then t**
else state-11-to-11-induction (**f**,
data-out,
addr-out,
trunc (DEFAULT-VISUAL-MEM-VALUE,
MACHINE-SIZE),
watch-dog (**f**, **t**, **f**, *data-out*, *addr-out*),
$n - 1$) **endif**

THEOREM: state-11-to-12-mem-wait-store
((*mar11* = nat-to-bv (11, NXSZ))
∧ (*no-store* = **f**)
∧ (¬ b-direct-reg-b (*i-reg*)))
→ (big-machine (*mar11*,
f,

t,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog-history,
list-of-n-plus-1-dtack-reset-off (*n*))
= list (nat-to-bv (11, NXSZ),
f,
t,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, 16),
real-mem,
watch-dog (**f**, **t**, **f**, *data-out*, *addr-out*)))

EVENT: Disable state-11-to-12-mem-wait-store.

THEOREM: state-11-to-12-mem-dtack-store
(sizep (*addr-out*, MACHINE-SIZE)
∧ (*mar11* = nat-to-bv (11, NXSZ))
∧ (*no-store* = **f**)

$\wedge$ (¬ b-direct-reg-b (*i-reg*)))
$\rightarrow$ (big-machine (*mar11*,
f,
t,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog (**f**, **t**, **f**, *data-out*, *addr-out*),
list (list (**t**, **f**), list (**t**, **f**)))
= list (nat-to-bv (12, NXSZ),
f,
t,
t,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, 16),
update-v-nth (**t**, *addr-out*, *real-mem*, *data-out*),
watch-dog (**f**, **t**, **t**, *data-out*, *addr-out*)))

EVENT: Disable state-11-to-12-mem-dtack-store.

THEOREM: state-10-to-12-reg

```
((mar10 = nat-to-bv (10, NXSZ))
 ∧   boolp (no-store)
 ∧   boolp (c-flag)
 ∧   boolp (v-flag)
 ∧   boolp (n-flag)
 ∧   boolp (z-flag)
 ∧   sizep (data-out, MACHINE-SIZE)
 ∧   sizep (a-reg, MACHINE-SIZE)
 ∧   ramp (reg-file, MACHINE-SIZE, 8)
 ∧   b-direct-reg-b (i-reg))
→    (big-machine (mar10,
                   f,
                   f,
                   dtack,
                   f,
                   no-store,
                   data-out,
                   reg-file,
                   addr-out,
                   c-flag,
                   v-flag,
                   z-flag,
                   n-flag,
                   a-reg,
                   b-reg,
                   i-reg,
                   visual-mem,
                   real-mem,
                   watch-dog-history,
                   list (list (f, f), list (f, f)))
      =   list (nat-to-bv (12, NXSZ),
                f,
                f,
                f,
                f,
                ¬ b-store-alu-result (c-flag, v-flag, z-flag, n-flag, i-reg),
                bv (bv-alu-cv (a-reg, b-reg, c-flag, bv-alu-op-code (i-reg))),
                update-v-nth (b-store-alu-result (c-flag,
                                                  v-flag,
                                                  z-flag,
                                                  n-flag,
                                                  i-reg),
                              bv-oprd-b (i-reg),
                              reg-file,
```

bv (bv-alu-cv (*a-reg*,
b-reg,
c-flag,
bv-alu-op-code (*i-reg*)))),
addr-out,
alu-c-flag (*i-reg*, *a-reg*, *b-reg*, *c-flag*),
alu-v-flag (*i-reg*, *a-reg*, *b-reg*, *c-flag*, *v-flag*),
alu-z-flag (*i-reg*, *a-reg*, *b-reg*, *c-flag*, *z-flag*),
alu-n-flag (*i-reg*, *a-reg*, *b-reg*, *c-flag*, *n-flag*),
a-reg,
b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, 16),
real-mem,
watch-dog (**f**,
f,
f,
bv (bv-alu-cv (*a-reg*,
b-reg,
c-flag,
bv-alu-op-code (*i-reg*))),
addr-out)))

EVENT: Disable state-10-to-12-reg.

THEOREM: state-10-to-12-mem-no-store
((*mar10* = nat-to-bv (10, NXSZ))
∧ boolp (*c-flag*)
∧ boolp (*v-flag*)
∧ boolp (*n-flag*)
∧ boolp (*z-flag*)
∧ sizep (*data-out*, MACHINE-SIZE)
∧ sizep (*a-reg*, MACHINE-SIZE)
∧ (¬ b-direct-reg-b (*i-reg*))
∧ (¬ b-store-alu-result (*c-flag*, *v-flag*, *z-flag*, *n-flag*, *i-reg*)))
→ (big-machine (*mar10*,
f,
f,
dtack,
f,
no-store,
data-out,
reg-file,
addr-out,

c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog-history,
list (list (**f**, **f**), list (**f**, **f**)))

= list (nat-to-bv (12, NXSZ),
f,
f,
f,
f,
¬ b-store-alu-result (*c-flag*, *v-flag*, *z-flag*, *n-flag*, *i-reg*),
bv (bv-alu-cv (*a-reg*, *b-reg*, *c-flag*, bv-alu-op-code (*i-reg*))),
reg-file,
addr-out,
alu-c-flag (*i-reg*, *a-reg*, *b-reg*, *c-flag*),
alu-v-flag (*i-reg*, *a-reg*, *b-reg*, *c-flag*, *v-flag*),
alu-z-flag (*i-reg*, *a-reg*, *b-reg*, *c-flag*, *z-flag*),
alu-n-flag (*i-reg*, *a-reg*, *b-reg*, *c-flag*, *n-flag*),
a-reg,
b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, 16),
real-mem,
watch-dog (**f**,
f,
f,
bv (bv-alu-cv (*a-reg*,
b-reg,
c-flag,
bv-alu-op-code (*i-reg*))),
addr-out)))

EVENT: Disable state-10-to-12-mem-no-store.

THEOREM: state-10-to-12-mem-store
((*mar10* = nat-to-bv (10, NXSZ))
∧ boolp (*no-store*)
∧ boolp (*c-flag*)

$\wedge$ boolp (*v-flag*)
$\wedge$ boolp (*n-flag*)
$\wedge$ boolp (*z-flag*)
$\wedge$ sizep (*data-out*, MACHINE-SIZE)
$\wedge$ sizep (*addr-out*, MACHINE-SIZE)
$\wedge$ sizep (*a-reg*, MACHINE-SIZE)
$\wedge$ ($\neg$ b-direct-reg-b (*i-reg*))
$\wedge$ b-store-alu-result (*c-flag*, *v-flag*, *z-flag*, *n-flag*, *i-reg*))
$\rightarrow$ (big-machine (*mar10*,
f,
f,
dtack,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog-history,
append (list (list (**f**, **f**)),
append (list (list (**f**, **f**)),
append (list-of-n-plus-1-dtack-reset-off (*n*),
list (list (**t**, **f**), list (**t**, **f**))))))
= list (nat-to-bv (12, NXSZ),
f,
t,
t,
f,
$\neg$ b-store-alu-result (*c-flag*, *v-flag*, *z-flag*, *n-flag*, *i-reg*),
bv (bv-alu-cv (*a-reg*, *b-reg*, *c-flag*, bv-alu-op-code (*i-reg*))),
reg-file,
addr-out,
alu-c-flag (*i-reg*, *a-reg*, *b-reg*, *c-flag*),
alu-v-flag (*i-reg*, *a-reg*, *b-reg*, *c-flag*, *v-flag*),
alu-z-flag (*i-reg*, *a-reg*, *b-reg*, *c-flag*, *z-flag*),
alu-n-flag (*i-reg*, *a-reg*, *b-reg*, *c-flag*, *n-flag*),

a-reg,
b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, 16),
update-v-nth (**t**,
addr-out,
real-mem,
bv (bv-alu-cv (*a-reg*,
b-reg,
c-flag,
bv-alu-op-code (*i-reg*)))),
watch-dog (**f**,
t,
t,
bv (bv-alu-cv (*a-reg*,
b-reg,
c-flag,
bv-alu-op-code (*i-reg*))),
addr-out)))

EVENT: Disable state-10-to-12-mem-store.

DEFINITION:
alu-oracle (*i-reg*, *c-flag*, *v-flag*, *z-flag*, *n-flag*, *n*)
= **if** b-direct-reg-b (*i-reg*) **then** list (list (**f**, **f**), list (**f**, **f**))
elseif b-store-alu-result (*c-flag*, *v-flag*, *z-flag*, *n-flag*, *i-reg*)
then append (list (list (**f**, **f**)),
append (list (list (**f**, **f**)),
append (list-of-n-plus-1-dtack-reset-off (*n*),
list (list (**t**, **f**), list (**t**, **f**)))))
else list (list (**f**, **f**), list (**f**, **f**)) **endif**

THEOREM: update-nth-f
update-nth (**f**, *n*, *lst*, *val*) = *lst*

DEFINITION:
mem-store-flag (*i-reg*, *c-flag*, *v-flag*, *z-flag*, *n-flag*)
= **if** b-direct-reg-b (*i-reg*) **then f**
else b-store-alu-result (*c-flag*, *v-flag*, *z-flag*, *n-flag*, *i-reg*) **endif**

THEOREM: state-10-to-12
(boolp (*no-store*)
∧ boolp (*c-flag*)
∧ boolp (*v-flag*)

```
  ∧   boolp (n-flag)
  ∧   boolp (z-flag)
  ∧   sizep (data-out, MACHINE-SIZE)
  ∧   sizep (a-reg, MACHINE-SIZE)
  ∧   sizep (addr-out, MACHINE-SIZE)
  ∧   ramp (reg-file, MACHINE-SIZE, 8)
  ∧   (mar10 = nat-to-bv (10, NXSZ)))
→   (big-machine (mar10,
                  f,
                  f,
                  dtack,
                  f,
                  no-store,
                  data-out,
                  reg-file,
                  addr-out,
                  c-flag,
                  v-flag,
                  z-flag,
                  n-flag,
                  a-reg,
                  b-reg,
                  i-reg,
                  visual-mem,
                  real-mem,
                  watch-dog-history,
                  alu-oracle (i-reg, c-flag, v-flag, z-flag, n-flag, n))
      =   list (nat-to-bv (12, NXSZ),
                f,
                mem-store-flag (i-reg, c-flag, v-flag, z-flag, n-flag),
                mem-store-flag (i-reg, c-flag, v-flag, z-flag, n-flag),
                f,
                ¬ b-store-alu-result (c-flag, v-flag, z-flag, n-flag, i-reg),
                bv (bv-alu-cv (a-reg, b-reg, c-flag, bv-alu-op-code (i-reg))),
                update-v-nth (b-direct-reg-b (i-reg)
                                  ∧   b-store-alu-result (c-flag,
                                                          v-flag,
                                                          z-flag,
                                                          n-flag,
                                                          i-reg),
                                  bv-oprd-b (i-reg),
                                  reg-file,
                                  bv (bv-alu-cv (a-reg,
                                                 b-reg,
```

```
                                    c-flag,
                                    bv-alu-op-code (i-reg)))),
addr-out,
alu-c-flag (i-reg, a-reg, b-reg, c-flag),
alu-v-flag (i-reg, a-reg, b-reg, c-flag, v-flag),
alu-z-flag (i-reg, a-reg, b-reg, c-flag, z-flag),
alu-n-flag (i-reg, a-reg, b-reg, c-flag, n-flag),
a-reg,
b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, 16),
update-v-nth ((¬ b-direct-reg-b (i-reg))
              ∧   b-store-alu-result (c-flag,
                                      v-flag,
                                      z-flag,
                                      n-flag,
                                      i-reg),
              addr-out,
              real-mem,
              bv (bv-alu-cv (a-reg,
                             b-reg,
                             c-flag,
                             bv-alu-op-code (i-reg)))),
watch-dog (f,
           mem-store-flag (i-reg,
                           c-flag,
                           v-flag,
                           z-flag,
                           n-flag),
           mem-store-flag (i-reg,
                           c-flag,
                           v-flag,
                           z-flag,
                           n-flag),
           bv (bv-alu-cv (a-reg,
                          b-reg,
                          c-flag,
                          bv-alu-op-code (i-reg))),
           addr-out)))
```

EVENT: Disable state-10-to-12.

EVENT: Disable mem-store-flag.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                                ;;
;;                       Post-increment operations                                ;;
;;                                                                                ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: state-12-to-13

(ramp (*reg-file*, MACHINE-SIZE, 8)
∧ (truep (*mem-store-flag*) ∨ falsep (*mem-store-flag*))
∧ (*mar12* = nat-to-bv (12, NXSZ)))
→ (big-machine (*mar12*,
f,
mem-store-flag,
mem-store-flag,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog (**f**,
mem-store-flag,
mem-store-flag,
data-out,
addr-out),
list (list (*mem-store-flag*, **f**)))
= list (nat-to-bv (13, NXSZ),
f,
f,
mem-store-flag,
f,
no-store,
data-out,
update-v-nth (b-indirect-reg-a-inc (*i-reg*),

bv-oprd-a (*i-reg*),
reg-file,
v-nat-inc (v-nth (bv-oprd-a (*i-reg*), *reg-file*))),
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, 16),
real-mem,
watch-dog (**f**,
mem-store-flag,
mem-store-flag,
data-out,
addr-out)))

THEOREM: state-13-to-1
(ramp (*reg-file*, MACHINE-SIZE, 8) ∧ (*mar13* = nat-to-bv (13, NXSZ)))
→ (big-machine (*mar13*,
f,
f,
dtack,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog-history,
list (list (**f**, **f**)))
= list (nat-to-bv (1, NXSZ),
f,
f,

f,
f,
no-store,
data-out,
update-v-nth (b-indirect-reg-b-inc (*i-reg*),
bv-oprd-b (*i-reg*),
reg-file,
v-nat-inc (v-nth (bv-oprd-b (*i-reg*), *reg-file*))),
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, 16),
real-mem,
watch-dog (**f**, **f**, **f**, *data-out*, *addr-out*)))

DEFINITION:
reg-file-post-inc (*i-reg*, *reg-file*)
= update-v-nth (b-indirect-reg-b-inc (*i-reg*),
bv-oprd-b (*i-reg*),
update-v-nth (b-indirect-reg-a-inc (*i-reg*),
bv-oprd-a (*i-reg*),
reg-file,
v-nat-inc (v-nth (bv-oprd-a (*i-reg*), *reg-file*))),
v-nat-inc (v-nth (bv-oprd-b (*i-reg*),
update-v-nth (b-indirect-reg-a-inc (*i-reg*),
bv-oprd-a (*i-reg*),
reg-file,
v-nat-inc (v-nth (bv-oprd-a (*i-reg*),
reg-file))))))

THEOREM: state-12-to-1
(ramp (*reg-file*, MACHINE-SIZE, 8)
∧ (truep (*mem-store-flag*) ∨ falsep (*mem-store-flag*))
∧ (*mar12* = nat-to-bv (12, NXSZ)))
→ (big-machine (*mar12*,
f,
mem-store-flag,
mem-store-flag,
f,

no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog (**f**,
mem-store-flag,
mem-store-flag,
data-out,
addr-out),
list (list (*mem-store-flag*, **f**), list (**f**, **f**)))

= list (nat-to-bv (`1`, NXSZ),
f,
f,
f,
f,
no-store,
data-out,
reg-file-post-inc (*i-reg*, *reg-file*),
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
trunc (DEFAULT-VISUAL-MEM-VALUE, `16`),
real-mem,
watch-dog (**f**, **f**, **f**, *data-out*, *addr-out*)))

EVENT: Disable state-12-to-1.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;;                    Pasting together the steps                              ;;
```

```
;;                                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
; In state-4-to-7 we rewrite to something of the form (list ... (if a b c) ...).
; When applying big-machine-append the variable M is bound to the rewritten version
; of this form.  But the rewriter moves the if's out, producing
; (if a (list ... b ...) (list ... c ...)).  We then, in simplifying the rhs of
; big-machine-append, rewrite (cadr M) to get the new read.  We want it to rewrite to
; f but in fact it rewrites to (cadr (if a (list mar7 f ...) (list mar7 f ...))).
; Then, state-7-to-10 doesn't unify.  So we prove the following two facts.
; Because of these, we don't really care whether our state steppers return
; lists with if's inside or outside.
```

THEOREM: car-if
car (**if** *a* **then** *b*
else *c* **endif**)
= **if** *a* **then** car (*b*)
else car (*c*) **endif**

THEOREM: cdr-if
cdr (**if** *a* **then** *b*
else *c* **endif**)
= **if** *a* **then** cdr (*b*)
else cdr (*c*) **endif**

DEFINITION:
reg-file-after-a-b (*reg-file*, *real-mem*)
= fetch-oprd-b-reg-file (v-nth (nth (0, *reg-file*), *real-mem*),
fetch-oprd-a-reg-file (v-nth (nth (0, *reg-file*),
real-mem),
update-nth (**t**,
0,
reg-file,
v-nat-inc (nth (0,
reg-file)))))

DEFINITION:
addr-out-after-a-b (*reg-file*, *real-mem*)
= fetch-oprd-b-addr-out (v-nth (nth (0, *reg-file*), *real-mem*),
fetch-oprd-a-reg-file (v-nth (nth (0, *reg-file*),
real-mem),
update-nth (**t**,
0,

reg-file,
v-nat-inc (nth (0,
reg-file)))))

DEFINITION:
a-reg-after-a-b (*reg-file*, *real-mem*)
= **if** b-direct-reg-a (v-nth (nth (0, *reg-file*), *real-mem*))
then v-nth (bv-oprd-a (v-nth (nth (0, *reg-file*), *real-mem*)),
fetch-oprd-a-reg-file (v-nth (nth (0, *reg-file*), *real-mem*),
update-nth (**t**,
0,
reg-file,
v-nat-inc (nth (0, *reg-file*)))))
else v-nth (fetch-oprd-a-addr-out (v-nth (nth (0, *reg-file*),
real-mem),
update-nth (**t**,
0,
reg-file,
v-nat-inc (nth (0,
reg-file)))),
real-mem) **endif**

DEFINITION:
b-reg-after-a-b (*reg-file*, *real-mem*)
= **if** b-direct-reg-b (v-nth (nth (0, *reg-file*), *real-mem*))
then v-nth (bv-oprd-b (v-nth (nth (0, *reg-file*), *real-mem*)),
fetch-oprd-b-reg-file (v-nth (nth (0, *reg-file*), *real-mem*),
fetch-oprd-a-reg-file (v-nth (nth (0,
reg-file),
real-mem),
update-nth (**t**,
0,
reg-file,
v-nat-inc (nth (0,
reg-file))))))
else v-nth (fetch-oprd-b-addr-out (v-nth (nth (0, *reg-file*),
real-mem),
fetch-oprd-a-reg-file (v-nth (nth (0,
reg-file),
real-mem),
update-nth (**t**,
0,
reg-file,
v-nat-inc (nth (0,

reg-file))))),
real-mem) **endif**

DEFINITION:
visual-mem-after-a-b (*reg-file*, *real-mem*)
= **if** b-direct-reg-b (v-nth (nth (0, *reg-file*), *real-mem*))
then trunc (DEFAULT-VISUAL-MEM-VALUE, 16)
else v-nth (fetch-oprd-b-addr-out (v-nth (nth (0, *reg-file*),
real-mem),
fetch-oprd-a-reg-file (v-nth (nth (0,
reg-file),
real-mem),
update-nth (**t**,
0,
reg-file,
v-nat-inc (nth (0,
reg-file))))),
real-mem) **endif**

DEFINITION:
dtack-after-a-b (*reg-file*, *real-mem*)
= (¬ b-direct-reg-b (v-nth (nth (0, *reg-file*), *real-mem*)))

DEFINITION:
a-b-oracle (*reg-file*, *real-mem*, *i*, *n*, *m*)
= append (fetch-ir-oracle (*i*),
append (fetch-oprd-a-oracle (v-nth (nth (0, *reg-file*), *real-mem*), *n*),
fetch-oprd-b-oracle (v-nth (nth (0, *reg-file*), *real-mem*), *m*)))

THEOREM: state-1-to-10
(ramp (*reg-file*, MACHINE-SIZE, 8)
∧ sizep (*addr-out*, MACHINE-SIZE)
∧ ramp (*real-mem*, MACHINE-SIZE, exp (2, MACHINE-SIZE))
∧ (*mar1* = nat-to-bv (1, NXSZ)))
→ (big-machine (*mar1*,
f,
f,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,

z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog,
a-b-oracle (*reg-file*, *real-mem*, *i*, *n*, *m*))
= list (nat-to-bv (10, NXSZ),
f,
f,
dtack-after-a-b (*reg-file*, *real-mem*),
f,
no-store,
data-out,
reg-file-after-a-b (*reg-file*, *real-mem*),
addr-out-after-a-b (*reg-file*, *real-mem*),
c-flag,
v-flag,
z-flag,
n-flag,
a-reg-after-a-b (*reg-file*, *real-mem*),
b-reg-after-a-b (*reg-file*, *real-mem*),
v-nth (nth (0, *reg-file*), *real-mem*),
visual-mem-after-a-b (*reg-file*, *real-mem*),
real-mem,
watch-dog (dtack-after-a-b (*reg-file*, *real-mem*),
f,
dtack-after-a-b (*reg-file*, *real-mem*),
data-out,
addr-out-after-a-b (*reg-file*, *real-mem*))))

DEFINITION:
reg-file-alu (*i-reg*, *c-flag*, *v-flag*, *z-flag*, *n-flag*, *reg-file*, *a-reg*, *b-reg*)
= update-v-nth (b-direct-reg-b (*i-reg*)
∧ b-store-alu-result (*c-flag*,
v-flag,
z-flag,
n-flag,
i-reg),
bv-oprd-b (*i-reg*),
reg-file,
bv (bv-alu-cv (*a-reg*, *b-reg*, *c-flag*, bv-alu-op-code (*i-reg*))))

```
DEFINITION:
reg-file-from-alu-thru-post-inc (i-reg,
                                 c-flag,
                                 v-flag,
                                 z-flag,
                                 n-flag,
                                 reg-file,
                                 a-reg,
                                 b-reg)
=  update-v-nth (b-indirect-reg-b-inc (i-reg),
                 bv-oprd-b (i-reg),
                 update-v-nth (b-indirect-reg-a-inc (i-reg),
                               bv-oprd-a (i-reg),
                               reg-file-alu (i-reg,
                                             c-flag,
                                             v-flag,
                                             z-flag,
                                             n-flag,
                                             reg-file,
                                             a-reg,
                                             b-reg),
                               v-nat-inc (v-nth (bv-oprd-a (i-reg),
                                                 reg-file-alu (i-reg,
                                                               c-flag,
                                                               v-flag,
                                                               z-flag,
                                                               n-flag,
                                                               reg-file,
                                                               a-reg,
                                                               b-reg)))),
                 v-nat-inc (v-nth (bv-oprd-b (i-reg),
                                   update-v-nth (b-indirect-reg-a-inc (i-reg),
                                                 bv-oprd-a (i-reg),
                                                 reg-file-alu (i-reg,
                                                               c-flag,
                                                               v-flag,
                                                               z-flag,
                                                               n-flag,
                                                               reg-file,
                                                               a-reg,
                                                               b-reg),
                                                 v-nat-inc (v-nth (bv-oprd-a (i-reg),
                                                                   reg-file-alu (i-reg,
                                                                                 c-flag,
```

v-flag,
z-flag,
n-flag,
reg-file,
a-reg,
b-reg)))))))

DEFINITION:
real-mem-thru-post-inc (*i-reg*,
c-flag,
v-flag,
z-flag,
n-flag,
addr-out,
real-mem,
a-reg,
b-reg)
= update-v-nth ((¬ b-direct-reg-b (*i-reg*))
∧ b-store-alu-result (*c-flag*,
v-flag,
z-flag,
n-flag,
i-reg),
addr-out,
real-mem,
bv (bv-alu-cv (*a-reg*, *b-reg*, *c-flag*, bv-alu-op-code (*i-reg*))))

DEFINITION:
alu-post-inc-oracle (*i-reg*, *c-flag*, *v-flag*, *z-flag*, *n-flag*, *n*)
= append (alu-oracle (*i-reg*, *c-flag*, *v-flag*, *z-flag*, *n-flag*, *n*),
list (list (mem-store-flag (*i-reg*, *c-flag*, *v-flag*, *z-flag*, *n-flag*), **f**),
list (**f**, **f**)))

THEOREM: ramp-if
ramp (**if** *c* **then** *a*
else *b* **endif**,
i,
j)
= **if** *c* **then** ramp (*a*, *i*, *j*)
else ramp (*b*, *i*, *j*) **endif**

THEOREM: ramp-update-nth
(ramp (*reg-file*, *width*, *number*)
∧ sizep (*value*, *width*)
∧ boolp (*boolean*)

```
  ∧   (number ≮ place))
→   ramp (update-nth (boolean, place, reg-file, value), width, number)

THEOREM: state-10-to-1
 (boolp (no-store)
  ∧   boolp (c-flag)
  ∧   boolp (v-flag)
  ∧   boolp (n-flag)
  ∧   boolp (z-flag)
  ∧   sizep (data-out, MACHINE-SIZE)
  ∧   sizep (a-reg, MACHINE-SIZE)
  ∧   sizep (addr-out, MACHINE-SIZE)
  ∧   ramp (reg-file, MACHINE-SIZE, 8)
  ∧   (mar10 = nat-to-bv (10, NXSZ)))
→   (big-machine (mar10,
                  f,
                  f,
                  dtack,
                  f,
                  no-store,
                  data-out,
                  reg-file,
                  addr-out,
                  c-flag,
                  v-flag,
                  z-flag,
                  n-flag,
                  a-reg,
                  b-reg,
                  i-reg,
                  visual-mem,
                  real-mem,
                  watch-dog-history,
                  alu-post-inc-oracle (i-reg, c-flag, v-flag, z-flag, n-flag, n))
      =   list (nat-to-bv (1, NXSZ),
                f,
                f,
                f,
                f,
                ¬ b-store-alu-result (c-flag, v-flag, z-flag, n-flag, i-reg),
                bv (bv-alu-cv (a-reg, b-reg, c-flag, bv-alu-op-code (i-reg))),
                reg-file-from-alu-thru-post-inc (i-reg,
                                                 c-flag,
                                                 v-flag,
```

```
                                                  z-flag,
                                                  n-flag,
                                                  reg-file,
                                                  a-reg,
                                                  b-reg),
            addr-out,
            alu-c-flag (i-reg, a-reg, b-reg, c-flag),
            alu-v-flag (i-reg, a-reg, b-reg, c-flag, v-flag),
            alu-z-flag (i-reg, a-reg, b-reg, c-flag, z-flag),
            alu-n-flag (i-reg, a-reg, b-reg, c-flag, n-flag),
            a-reg,
            b-reg,
            i-reg,
            trunc (DEFAULT-VISUAL-MEM-VALUE, 16),
            real-mem-thru-post-inc (i-reg,
                                    c-flag,
                                    v-flag,
                                    z-flag,
                                    n-flag,
                                    addr-out,
                                    real-mem,
                                    a-reg,
                                    b-reg),
            watch-dog (f,
                       f,
                       f,
                       bv (bv-alu-cv (a-reg,
                                      b-reg,
                                      c-flag,
                                      bv-alu-op-code (i-reg))),
                       addr-out)))
```

EVENT: Disable state-10-to-1.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;;           The lemmas that follow culminate in lemma STATE-1-to-1           ;;
;;           which is a statement of what FM8501 does upon executing          ;;
;;           an instruction.  FM8501 executes one memory based instruction    ;;
;;           by starting in a state where the MAR is "1", executing some      ;;
;;           of micro-states eventually taking it back to a place where       ;;
;;           MAR is equal to "1".                                             ;;
;;                                                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: sizep-a-reg-after-a-b
(ramp (*reg-file*, MACHINE-SIZE, 8)
 ∧ ramp (*real-mem*, MACHINE-SIZE, exp (2, MACHINE-SIZE)))
→ (size (a-reg-after-a-b (*reg-file*, *real-mem*)) = MACHINE-SIZE)

THEOREM: sizep-addr-out-after-a-b
ramp (*reg-file*, MACHINE-SIZE, 8)
→ (size (addr-out-after-a-b (*reg-file*, *real-mem*)) = MACHINE-SIZE)

THEOREM: ramp-reg-file-after-a-b
((*m* = MACHINE-SIZE) ∧ ramp (*reg-file*, MACHINE-SIZE, 8))
→ ramp (reg-file-after-a-b (*reg-file*, *real-mem*), *m*, 8)

DEFINITION:
state-1-to-1-oracle (*reg-file*,
real-mem,
c-flag,
v-flag,
z-flag,
n-flag,
ir-wait,
a-wait,
b-wait,
store-wait)
= append (a-b-oracle (*reg-file*, *real-mem*, *ir-wait*, *a-wait*, *b-wait*),
alu-post-inc-oracle (v-nth (nth (0, *reg-file*), *real-mem*),
c-flag,
v-flag,
z-flag,
n-flag,
store-wait))

THEOREM: state-1-to-1
(boolp (*no-store*)
 ∧ boolp (*c-flag*)
 ∧ boolp (*v-flag*)
 ∧ boolp (*n-flag*)
 ∧ boolp (*z-flag*)
 ∧ ramp (*reg-file*, MACHINE-SIZE, 8)
 ∧ sizep (*data-out*, MACHINE-SIZE)
 ∧ sizep (*addr-out*, MACHINE-SIZE)
 ∧ ramp (*real-mem*, MACHINE-SIZE, exp (2, MACHINE-SIZE))

```
 ∧   (mar1 = nat-to-bv (1, NXSZ)))
→   (big-machine (mar1,
                  f,
                  f,
                  f,
                  f,
                  no-store,
                  data-out,
                  reg-file,
                  addr-out,
                  c-flag,
                  v-flag,
                  z-flag,
                  n-flag,
                  a-reg,
                  b-reg,
                  i-reg,
                  visual-mem,
                  real-mem,
                  watch-dog,
                  state-1-to-1-oracle (reg-file,
                                       real-mem,
                                       c-flag,
                                       v-flag,
                                       z-flag,
                                       n-flag,
                                       ir-wait,
                                       a-wait,
                                       b-wait,
                                       store-wait))
    =   list (nat-to-bv (1, NXSZ),
              f,
              f,
              f,
              f,
              ¬ b-store-alu-result (c-flag,
                                    v-flag,
                                    z-flag,
                                    n-flag,
                                    v-nth (nth (0, reg-file), real-mem)),
              bv (bv-alu-cv (a-reg-after-a-b (reg-file, real-mem),
                             b-reg-after-a-b (reg-file, real-mem),
                             c-flag,
                             bv-alu-op-code (v-nth (nth (0, reg-file),
```

real-mem))))),
reg-file-from-alu-thru-post-inc (v-nth (nth (0, *reg-file*),
real-mem),
c-flag,
v-flag,
z-flag,
n-flag,
reg-file-after-a-b (*reg-file*,
real-mem),
a-reg-after-a-b (*reg-file*,
real-mem),
b-reg-after-a-b (*reg-file*,
real-mem)),
addr-out-after-a-b (*reg-file*, *real-mem*),
alu-c-flag (v-nth (nth (0, *reg-file*), *real-mem*),
a-reg-after-a-b (*reg-file*, *real-mem*),
b-reg-after-a-b (*reg-file*, *real-mem*),
c-flag),
alu-v-flag (v-nth (nth (0, *reg-file*), *real-mem*),
a-reg-after-a-b (*reg-file*, *real-mem*),
b-reg-after-a-b (*reg-file*, *real-mem*),
c-flag,
v-flag),
alu-z-flag (v-nth (nth (0, *reg-file*), *real-mem*),
a-reg-after-a-b (*reg-file*, *real-mem*),
b-reg-after-a-b (*reg-file*, *real-mem*),
c-flag,
z-flag),
alu-n-flag (v-nth (nth (0, *reg-file*), *real-mem*),
a-reg-after-a-b (*reg-file*, *real-mem*),
b-reg-after-a-b (*reg-file*, *real-mem*),
c-flag,
n-flag),
a-reg-after-a-b (*reg-file*, *real-mem*),
b-reg-after-a-b (*reg-file*, *real-mem*),
v-nth (nth (0, *reg-file*), *real-mem*),
trunc (DEFAULT-VISUAL-MEM-VALUE, 16),
real-mem-thru-post-inc (v-nth (nth (0, *reg-file*), *real-mem*),
c-flag,
v-flag,
z-flag,
n-flag,
addr-out-after-a-b (*reg-file*,
real-mem),

```
                                  real-mem,
                                  a-reg-after-a-b (reg-file, real-mem),
                                  b-reg-after-a-b (reg-file, real-mem)),
            watch-dog (f,
                       f,
                       f,
                       bv (bv-alu-cv (a-reg-after-a-b (reg-file, real-mem),
                                      b-reg-after-a-b (reg-file, real-mem),
                                      c-flag,
                                      bv-alu-op-code (v-nth (nth (0,
                                                                  reg-file),
                                                             real-mem)))),
                       addr-out-after-a-b (reg-file, real-mem))))
```

EVENT: Disable state-1-to-1.

THEOREM: nth-update-nth
nth (i, update-nth (c, j, lst, x))
= **if** truep (c) $\wedge$ (fix (i) = fix (j)) $\wedge$ (j < length (lst)) **then** x
else nth (i, lst) **endif**

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                              ;;
;;          Software specification of FM8501.                                   ;;
;;                                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                              ;;
;;          Helping definitions for the specification of the software           ;;
;;          machine.  Function SOFT is the actual definition defining           ;;
;;          the operation of the FM8501.                                        ;;
;;                                                                              ;;
;;          When programming the FM8501 the user see eight general purpose      ;;
;;          registers (register file), four flag bits (carry, overflow,         ;;
;;          negative, and zero), and the memory.  Instruction execution         ;;
;;          is accomplished by reading the memory value pointed to by the       ;;
;;          program counter (register 0), and interpreting this memory word     ;;
;;          as an instruction to be executed.  The execution of an instruction  ;;
;;          may modify all visable registers.  The flag registers may be        ;;
;;          updated only once per instruction.  The memory may be updated       ;;
;;          only once per instruction.  The register file may be updated a      ;;
;;          maximum of four times in six opportunities.  The first update to    ;;
```

```
;;          the register file, which always occurs, is the incrementing of      ;;
;;          the program counter.  Next comes the possible pre-decrement         ;;
;;          operation of both operands.  The result of the ALU may be next      ;;
;;          stored into the register file.  Lastly, there may be two            ;;
;;          post-increment operations.  An operand cannot be both               ;;
;;          pre-decremented and post-incremented.                               ;;
;;                                                                              ;;
;;          The functions below generate the next state for the memory and      ;;
;;          the register file given an instruction.  These are grouped          ;;
;;          together into an interpreter called SOFT.  SOFT executed one        ;;
;;          instruction per recursive call.                                     ;;
;;                                                                              ;;
;;          The next state for the register file (REG-FILE) is created by       ;;
;;          the composition of several functions.  Each function describes      ;;
;;          the state of the register file at some point through the execution  ;;
;;          of an instruction.  Below are described the operation of the        ;;
;;          functions that update the register file.  All of these functions    ;;
;;          return a complete register file, which they may or may not of       ;;
;;          updated.                                                            ;;
;;                                                                              ;;
;;          reg-file-after-pc-increment --                                      ;;
;;               increments the PC (register 0)                                 ;;
;;                                                                              ;;
;;          reg-file-after-oprd-a-pre-decrement --                              ;;
;;               performs a pre-decrement operation for operand A if required,  ;;
;;               and included is the reg-file-after-pc-increment operation      ;;
;;                                                                              ;;
;;          reg-file-after-oprd-b-pre-decrement --                              ;;
;;               performs a pre-decrement operation for operand B if required,  ;;
;;               and included is the reg-file-after-oprd-a-pre-decrement        ;;
;;               operation                                                      ;;
;;                                                                              ;;
;;          reg-file-after-alu-write --                                         ;;
;;               writes the output of the ALU to the reg-file if required,      ;;
;;               and included is the reg-file-after-oprd-b-pre-decrement        ;;
;;               operation                                                      ;;
;;                                                                              ;;
;;          reg-file-after-oprd-a-post-increment --                             ;;
;;               performs a post-increment operation for operand A if required, ;;
;;               and included is the reg-file-after-alu-write operation         ;;
;;                                                                              ;;
;;          reg-file-after-oprd-b-post-increment --                             ;;
;;               performs a post-increment operation for operand B if required, ;;
;;               and included is the reg-file-after-oprd-a-post-increment       ;;
```

```
;;               operation                                                  ;;
;;                                                                          ;;
;;          The ALU requires three operands so its result can be calculated.  ;;
;;          The C-FLAG is readily available, but the A and B value for the    ;;
;;          ALU must be generated under instruction control.  The A value is  ;;
;;          fetched either from the register file or the memory, but not until ;;
;;          both the PC has been incremented and the operand A pre-decrement  ;;
;;          performed.  The function A-VALUE-FOR-ALU-AFTER-OPRD-A-PRE-DECREMENT ;;
;;          generates the appropriate value.  The B value for the ALU is      ;;
;;          fetched from either the register file or memory, but not until the ;;
;;          PC has been incremented and the possible operand A pre-decrement  ;;
;;          and operand B pre-decrement operations performed.  The function   ;;
;;          B-VALUE-FOR-ALU-AFTER-OPRD-B-PRE-DECREMENT generates the correct  ;;
;;          operand B value.                                                  ;;
;;                                                                          ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;  This function increments the Program Counter (PC) by one.  The PC is
;  stored in register 0 in the register file.
```

DEFINITION:
reg-file-after-pc-increment (*reg-file*)
= update-nth (**t**, 0, *reg-file*, v-nat-inc (nth (0, *reg-file*)))

```
;  This function extracts the current instruction from memory.
```

DEFINITION:
current-instruction (*reg-file*, *real-mem*) = v-nth (nth (0, *reg-file*), *real-mem*)

```
;  The result of this function are the contents of the register file after
;  the possible pre-decrement for operand A and the increment of PC.
```

DEFINITION:
reg-file-after-oprd-a-pre-decrement (*reg-file*, *real-mem*)
= update-v-nth (b-indirect-reg-a-dec (current-instruction (*reg-file*, *real-mem*)),
bv-oprd-a (current-instruction (*reg-file*, *real-mem*)),
reg-file-after-pc-increment (*reg-file*),
v-nat-dec (v-nth (bv-oprd-a (current-instruction (*reg-file*, *real-mem*)),
reg-file-after-pc-increment (*reg-file*))))

```
;  The result of this function are the contents of the register file after
;  the possible pre-decrement for operand B, the possible pre-decrement for
;  operand A and the increment of PC.
```

DEFINITION:
reg-file-after-oprd-b-pre-decrement (*reg-file*, *real-mem*)
= update-v-nth (b-indirect-reg-b-dec (current-instruction (*reg-file*, *real-mem*)),
bv-oprd-b (current-instruction (*reg-file*, *real-mem*)),
reg-file-after-oprd-a-pre-decrement (*reg-file*, *real-mem*),
v-nat-dec (v-nth (bv-oprd-b (current-instruction (*reg-file*, *real-mem*)),
reg-file-after-oprd-a-pre-decrement (*reg-file*, *real-mem*))))

```
;  This function returns operand A for use in the ALU after the PC has been
;  incremented and the possible operand A pre-decrement has occured.
```

DEFINITION:
a-value-for-alu-after-oprd-a-pre-decrement (*reg-file*, *real-mem*)
= **if** b-direct-reg-a (current-instruction (*reg-file*, *real-mem*))
then v-nth (bv-oprd-a (current-instruction (*reg-file*, *real-mem*)),
reg-file-after-pc-increment (*reg-file*))
elseif b-indirect-reg-a-dec (current-instruction (*reg-file*, *real-mem*))
then v-nth (v-nat-dec (v-nth (bv-oprd-a (current-instruction (*reg-file*, *real-mem*)),
reg-file-after-pc-increment (*reg-file*))),
real-mem)
else v-nth (v-nth (bv-oprd-a (current-instruction (*reg-file*, *real-mem*)),
reg-file-after-pc-increment (*reg-file*)),
real-mem) **endif**

```
;  This function returns operand B for use in the ALU after the PC has been
;  incremented, the possible operand A pre-decrement has occured, and the
;  possible operand B pre-decrement has occured.
```

DEFINITION:
b-value-for-alu-after-oprd-b-pre-decrement (*reg-file*, *real-mem*)
= **if** b-direct-reg-b (current-instruction (*reg-file*, *real-mem*))
then v-nth (bv-oprd-b (current-instruction (*reg-file*, *real-mem*)),
reg-file-after-oprd-a-pre-decrement (*reg-file*, *real-mem*))

elseif b-indirect-reg-b-dec (current-instruction (*reg-file*, *real-mem*))
then v-nth (v-nat-dec (v-nth (bv-oprd-b (current-instruction (*reg-file*, *real-mem*)), reg-file-after-oprd-a-pre-decrement (*reg-file*, *real-mem*))), *real-mem*)
else v-nth (v-nth (bv-oprd-b (current-instruction (*reg-file*, *real-mem*)), reg-file-after-oprd-a-pre-decrement (*reg-file*, *real-mem*)), *real-mem*) **endif**

```
;  This function returns the result of the ALU on operand A, operand B, and
;  the carry flag.
```

DEFINITION:
bv-alu-cv-results (*reg-file*, *real-mem*, *c-flag*)
= bv-alu-cv (a-value-for-alu-after-oprd-a-pre-decrement (*reg-file*, *real-mem*), b-value-for-alu-after-oprd-b-pre-decrement (*reg-file*, *real-mem*), *c-flag*, bv-alu-op-code (current-instruction (*reg-file*, *real-mem*)))

```
;  This function returns the result of a possible storing of the ALU result
;  into the register file.  Note that this function takes into account the
;  possible changes that may have been inflicted on the register file by some
;  pre-decrement operations and the PC increment.
```

DEFINITION:
reg-file-after-alu-write (*reg-file*, *real-mem*, *c-flag*, *v-flag*, *z-flag*, *n-flag*)
= update-v-nth (b-store-alu-result-with-ifs (*c-flag*, *v-flag*, *z-flag*, *n-flag*, current-instruction (*reg-file*, *real-mem*))
∧ b-direct-reg-b (current-instruction (*reg-file*, *real-mem*)),
bv-oprd-b (current-instruction (*reg-file*, *real-mem*)),
reg-file-after-oprd-b-pre-decrement (*reg-file*, *real-mem*),
bv (bv-alu-cv-results (*reg-file*, *real-mem*, *c-flag*)))

```
;  This function returns the result of a possible storing of the ALU result
;  into the memory.
```

DEFINITION:
real-mem-after-alu-write (*reg-file*, *real-mem*, *c-flag*, *v-flag*, *z-flag*, *n-flag*)
= update-v-nth (b-store-alu-result-with-ifs (*c-flag*,
v-flag,
z-flag,
n-flag,
current-instruction (*reg-file*,
real-mem))
∧ (¬ b-direct-reg-b (current-instruction (*reg-file*,
real-mem))),
v-nth (bv-oprd-b (current-instruction (*reg-file*, *real-mem*)),
reg-file-after-oprd-b-pre-decrement (*reg-file*,
real-mem)),
real-mem,
bv (bv-alu-cv-results (*reg-file*, *real-mem*, *c-flag*)))

```
;  This function returns the result of a possible post-increment of operand A
;  after the possible storing of the ALU result, the pre-decrements of A and B,
;  and the PC increment.
```

DEFINITION:
reg-file-after-oprd-a-post-increment (*reg-file*,
real-mem,
c-flag,
v-flag,
z-flag,
n-flag)
= update-v-nth (b-indirect-reg-a-inc (current-instruction (*reg-file*,
real-mem)),
bv-oprd-a (current-instruction (*reg-file*, *real-mem*)),
reg-file-after-alu-write (*reg-file*,
real-mem,
c-flag,
v-flag,
z-flag,
n-flag),
v-nat-inc (v-nth (bv-oprd-a (current-instruction (*reg-file*,
real-mem)),
reg-file-after-alu-write (*reg-file*,
real-mem,

```
                                                     c-flag,
                                                     v-flag,
                                                     z-flag,
                                                     n-flag))))
```

```
;  This function returns the result of a possible post-increment of operand B
;  after the possible post-increment of A, the storing of the ALU result, etc.
```

DEFINITION:

```
reg-file-after-oprd-b-post-increment (reg-file,
                                      real-mem,
                                      c-flag,
                                      v-flag,
                                      z-flag,
                                      n-flag)
=   update-v-nth (b-indirect-reg-b-inc (current-instruction (reg-file,
                                                             real-mem)),
                  bv-oprd-b (current-instruction (reg-file, real-mem)),
                  reg-file-after-oprd-a-post-increment (reg-file,
                                                        real-mem,
                                                        c-flag,
                                                        v-flag,
                                                        z-flag,
                                                        n-flag),
                  v-nat-inc (v-nth (bv-oprd-b (current-instruction (reg-file,
                                                                    real-mem)),
                             reg-file-after-oprd-a-post-increment (reg-file,
                                                                   real-mem,
                                                                   c-flag,
                                                                   v-flag,
                                                                   z-flag,
                                                                   n-flag))))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                              ;;
;;          Function SOFT is the programmer's interpreter for FM8501.           ;;
;;          This function completely defines the effect of all instructions     ;;
;;          on the programmer visable state.                                    ;;
;;                                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;  This function returns the result of executing n instructions, where
;  n is the length of lst.
```

DEFINITION:
soft (*reg-file*, *real-mem*, *c-flag*, *v-flag*, *z-flag*, *n-flag*, *lst*)
= **if** *lst* $\simeq$ **nil then** list (*reg-file*, *real-mem*, *c-flag*, *v-flag*, *z-flag*, *n-flag*)
else soft (reg-file-after-oprd-b-post-increment (*reg-file*, *real-mem*, *c-flag*, *v-flag*, *z-flag*, *n-flag*),
real-mem-after-alu-write (*reg-file*, *real-mem*, *c-flag*, *v-flag*, *z-flag*, *n-flag*),
update-v (b-cc-set (current-instruction (*reg-file*, *real-mem*)), *c-flag*, c (bv-alu-cv-results (*reg-file*, *real-mem*, *c-flag*))),
update-v (b-cc-set (current-instruction (*reg-file*, *real-mem*)), *v-flag*, v (bv-alu-cv-results (*reg-file*, *real-mem*, *c-flag*))),
update-v (b-cc-set (current-instruction (*reg-file*, *real-mem*)), *z-flag*, bv-to-nat (bv (bv-alu-cv-results (*reg-file*, *real-mem*, *c-flag*))) $\simeq$ 0),
update-v (b-cc-set (current-instruction (*reg-file*, *real-mem*)), *n-flag*, negativep (bv-to-tc (bv (bv-alu-cv-results (*reg-file*, *real-mem*, *c-flag*))))),
cdr (*lst*)) **endif**

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                              ;;
;;          Function SOFT-RESET specifies what the programmer sees after        ;;
;;          reset has occured.  Note that SOFT-RESET calls the function         ;;
;;          SOFT.  After a reset, the software instruction interpreter is       ;;
```

```
;;          is the function SOFT.                                                ;;
;;                                                                               ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:
soft-reset (*reg-file*, *real-mem*, *c-flag*, *v-flag*, *z-flag*, *n-flag*, *oracle*)
= **if** *oracle* ≃ **nil**
then list (*reg-file*, *real-mem*, *c-flag*, *v-flag*, *z-flag*, *n-flag*)
else soft (update-nth (**t**, 0, *reg-file*, nat-to-bv (0, MACHINE-SIZE)),
real-mem,
c-flag,
v-flag,
z-flag,
n-flag,
cdr (*oracle*)) **endif**

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                               ;;
;;          This file contains the events leading up to and including            ;;
;;          the proof that an abstraction, containing the programmer             ;;
;;          visable registers, of BIG-MACHINE is equivalent to SOFT.             ;;
;;          This equivalence is the main result of this work; this result        ;;
;;          proves the correctness of a large piece of hardware with             ;;
;;          respect to a high-level description.                                 ;;
;;                                                                               ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                               ;;
;;          Help lemmas for proving the correctness of "soft-machine"            ;;
;;                                                                               ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: a-reg-after-a-b-is-a-value-for-alu-after-oprd-a-pre-decrement
(ramp (*reg-file*, MACHINE-SIZE, 8)
∧ ramp (*real-mem*, MACHINE-SIZE, exp (2, MACHINE-SIZE)))
→ (a-reg-after-a-b (*reg-file*, *real-mem*)
= a-value-for-alu-after-oprd-a-pre-decrement (*reg-file*, *real-mem*))

THEOREM: b-reg-after-a-b-is-b-value-for-alu-after-oprd-b-pre-decrement
(ramp (*reg-file*, MACHINE-SIZE, 8)
 ∧ ramp (*real-mem*, MACHINE-SIZE, exp (2, MACHINE-SIZE)))
→ (b-reg-after-a-b (*reg-file*, *real-mem*)
    = b-value-for-alu-after-oprd-b-pre-decrement (*reg-file*, *real-mem*))

THEOREM: hard-c-flag-is-soft-c-flag
(ramp (*reg-file*, MACHINE-SIZE, 8)
 ∧ ramp (*real-mem*, MACHINE-SIZE, exp (2, MACHINE-SIZE))
 ∧ boolp (*c-flag*))
→ (alu-c-flag (nth (bv-to-nat (nth (0, *reg-file*)), *real-mem*),
                a-value-for-alu-after-oprd-a-pre-decrement (*reg-file*,
                                                            real-mem),
                b-value-for-alu-after-oprd-b-pre-decrement (*reg-file*,
                                                            real-mem),
                c-flag)
    = update-v (b-cc-set (current-instruction (*reg-file*, *real-mem*)),
                 c-flag,
                 c (bv-alu-cv-results (*reg-file*, *real-mem*, *c-flag*))))

THEOREM: hard-v-flag-is-soft-v-flag
(ramp (*reg-file*, MACHINE-SIZE, 8)
 ∧ ramp (*real-mem*, MACHINE-SIZE, exp (2, MACHINE-SIZE))
 ∧ boolp (*v-flag*))
→ (alu-v-flag (nth (bv-to-nat (nth (0, *reg-file*)), *real-mem*),
                a-value-for-alu-after-oprd-a-pre-decrement (*reg-file*,
                                                            real-mem),
                b-value-for-alu-after-oprd-b-pre-decrement (*reg-file*,
                                                            real-mem),
                c-flag,
                v-flag)
    = update-v (b-cc-set (current-instruction (*reg-file*, *real-mem*)),
                 v-flag,
                 v (bv-alu-cv-results (*reg-file*, *real-mem*, *c-flag*))))

THEOREM: hard-z-flag-is-soft-z-flag
(ramp (*reg-file*, MACHINE-SIZE, 8)
 ∧ ramp (*real-mem*, MACHINE-SIZE, exp (2, MACHINE-SIZE))
 ∧ boolp (*z-flag*))
→ (alu-z-flag (nth (bv-to-nat (nth (0, *reg-file*)), *real-mem*),
                a-value-for-alu-after-oprd-a-pre-decrement (*reg-file*,
                                                            real-mem),
                b-value-for-alu-after-oprd-b-pre-decrement (*reg-file*,
                                                            real-mem),
                c-flag,

z-flag)
= update-v (b-cc-set (current-instruction (*reg-file*, *real-mem*)),
z-flag,
bv-to-nat (bv (bv-alu-cv-results (*reg-file*,
real-mem,
c-flag))) ≃ 0))

THEOREM: sizep-a-value-for-alu-after-oprd-a-pre-decrement
(ramp (*reg-file*, MACHINE-SIZE, 8)
∧ ramp (*real-mem*, MACHINE-SIZE, exp (2, MACHINE-SIZE)))
→ (size (a-value-for-alu-after-oprd-a-pre-decrement (*reg-file*, *real-mem*))
= MACHINE-SIZE)

THEOREM: hard-n-flag-is-soft-n-flag
(ramp (*reg-file*, MACHINE-SIZE, 8)
∧ ramp (*real-mem*, MACHINE-SIZE, exp (2, MACHINE-SIZE))
∧ boolp (*n-flag*))
→ (alu-n-flag (nth (bv-to-nat (nth (0, *reg-file*)), *real-mem*),
a-value-for-alu-after-oprd-a-pre-decrement (*reg-file*,
real-mem),
b-value-for-alu-after-oprd-b-pre-decrement (*reg-file*,
real-mem),
c-flag,
n-flag)
= update-v (b-cc-set (current-instruction (*reg-file*, *real-mem*)),
n-flag,
negativep (bv-to-tc (bv (bv-alu-cv-results (*reg-file*,
real-mem,
c-flag))))))

THEOREM: addr-out-after-a-b-is-v-nth-of-reg-file-after-oprd-b-pre-decrement
(ramp (*reg-file*, MACHINE-SIZE, 8)
∧ ramp (*real-mem*, MACHINE-SIZE, exp (2, MACHINE-SIZE)))
→ (addr-out-after-a-b (*reg-file*, *real-mem*)
= v-nth (bv-oprd-b (current-instruction (*reg-file*, *real-mem*)),
reg-file-after-oprd-b-pre-decrement (*reg-file*, *real-mem*)))

THEOREM: hard-real-mem-is-soft-real-mem
(ramp (*reg-file*, MACHINE-SIZE, 8)
∧ ramp (*real-mem*, MACHINE-SIZE, exp (2, MACHINE-SIZE)))
→ (real-mem-thru-post-inc (nth (bv-to-nat (nth (0, *reg-file*)), *real-mem*),
c-flag,
v-flag,
z-flag,
n-flag,

nth (bv-to-nat (bv-oprd-b (current-instruction (*reg-file*,
real-mem))),
reg-file-after-oprd-b-pre-decrement (*reg-file*,
real-mem)),
real-mem,
a-value-for-alu-after-oprd-a-pre-decrement (*reg-file*,
real-mem),
b-value-for-alu-after-oprd-b-pre-decrement (*reg-file*,
real-mem))
= real-mem-after-alu-write (*reg-file*,
real-mem,
c-flag,
v-flag,
z-flag,
n-flag))

THEOREM: reg-file-after-a-b-is-reg-file-after-oprd-b-pre-decrement
(ramp (*reg-file*, MACHINE-SIZE, 8)
∧ ramp (*real-mem*, MACHINE-SIZE, exp (2, MACHINE-SIZE)))
→ (reg-file-after-a-b (*reg-file*, *real-mem*)
= reg-file-after-oprd-b-pre-decrement (*reg-file*, *real-mem*))

THEOREM: hard-reg-file-is-soft-reg-file
(ramp (*reg-file*, MACHINE-SIZE, 8)
∧ ramp (*real-mem*, MACHINE-SIZE, exp (2, MACHINE-SIZE)))
→ (reg-file-from-alu-thru-post-inc (nth (bv-to-nat (nth (0, *reg-file*)),
real-mem),
c-flag,
v-flag,
z-flag,
n-flag,
reg-file-after-oprd-b-pre-decrement (*reg-file*,
real-mem),
a-value-for-alu-after-oprd-a-pre-decrement (*reg-file*,
real-mem),
b-value-for-alu-after-oprd-b-pre-decrement (*reg-file*,
real-mem))
= reg-file-after-oprd-b-post-increment (*reg-file*,
real-mem,
c-flag,
v-flag,
z-flag,
n-flag))

;;

```
;;                                                                          ;;
;;          Oracle generation function for BIG-MACHINE given the oracle     ;;
;;          for SOFT.  The SOFT machine oracle is a list, each element of   ;;
;;          the list contains four arbitrary values.  This values are "fixed"  ;;
;;          into numbers and used to construct a BIG-MACHINE oracle.  These ;;
;;          four-tuples describe the possible delay when BIG-MACHINE accesses  ;;
;;          memory.  The values are the number of waits to be inserted in   ;;
;;          memory accesses.  The four-tuples meaning is as follows:        ;;
;;                                                                          ;;
;;          (ir-wait fetch-oprd-a-wait fetch-oprd-b-wait store-oprd-b-wait)  ;;
;;                                                                          ;;
;;          ir-wait -- number of wait states for fetching an instruction    ;;
;;          fetch-oprd-a-wait -- number of wait states for fetching OPERAND-A  ;;
;;          fetch-oprd-b-wait -- number of wait states for fetching OPERAND-B  ;;
;;          store-oprd-b-wait -- number of wait states for storing OPERAND-B   ;;
;;                                                                          ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:

oracle (*reg-file*, *real-mem*, *c-flag*, *v-flag*, *z-flag*, *n-flag*, *lst*)

= **if** *lst* $\simeq$ **nil then nil**

else append (state-1-to-1-oracle (*reg-file*,
real-mem,
c-flag,
v-flag,
z-flag,
n-flag,
car (car (*lst*)),
cadr (car (*lst*)),
caddr (car (*lst*)),
cadddr (car (*lst*))),
oracle (reg-file-after-oprd-b-post-increment (*reg-file*,
real-mem,
c-flag,
v-flag,
z-flag,
n-flag),
real-mem-after-alu-write (*reg-file*,
real-mem,
c-flag,
v-flag,
z-flag,

n-flag),
update-v (b-cc-set (current-instruction (*reg-file*,
real-mem)),
c-flag,
c (bv-alu-cv-results (*reg-file*,
real-mem,
c-flag))),
update-v (b-cc-set (current-instruction (*reg-file*,
real-mem)),
v-flag,
v (bv-alu-cv-results (*reg-file*,
real-mem,
c-flag))),
update-v (b-cc-set (current-instruction (*reg-file*,
real-mem)),
z-flag,
bv-to-nat (bv (bv-alu-cv-results (*reg-file*,
real-mem,
c-flag))) ≃ 0),
update-v (b-cc-set (current-instruction (*reg-file*,
real-mem)),
n-flag,
negativep (bv-to-tc (bv (bv-alu-cv-results (*reg-file*,
real-mem,
c-flag))))),
cdr (*lst*))) **endif**

DEFINITION:
big-machine-induction (*no-store*,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog,
lst)
= **if** *lst* ≃ **nil then t**

```
else big-machine-induction (¬ b-store-alu-result (c-flag,
                                                  v-flag,
                                                  z-flag,
                                                  n-flag,
                                                  v-nth (nth (0,
                                                              reg-file),
                                                             real-mem)),
                                        bv (bv-alu-cv (a-reg-after-a-b (reg-file,
                                                                        real-mem),
                                                       b-reg-after-a-b (reg-file,
                                                                        real-mem),
                                                       c-flag,
                                                       bv-alu-op-code (v-nth (nth (0,
                                                                                   reg-file),
                                                                              real-mem)))),
                                        reg-file-from-alu-thru-post-inc (v-nth (nth (0,
                                                                                    reg-file),
                                                                               real-mem),
                                                                         c-flag,
                                                                         v-flag,
                                                                         z-flag,
                                                                         n-flag,
                                                                         reg-file-after-a-b (reg-file,
                                                                                             real-mem),
                                                                         a-reg-after-a-b (reg-file,
                                                                                          real-mem),
                                                                         b-reg-after-a-b (reg-file,
                                                                                          real-mem)),
                                        addr-out-after-a-b (reg-file, real-mem),
                                        alu-c-flag (v-nth (nth (0, reg-file),
                                                           real-mem),
                                                    a-reg-after-a-b (reg-file,
                                                                     real-mem),
                                                    b-reg-after-a-b (reg-file,
                                                                     real-mem),
                                                    c-flag),
                                        alu-v-flag (v-nth (nth (0, reg-file),
                                                           real-mem),
                                                    a-reg-after-a-b (reg-file,
                                                                     real-mem),
                                                    b-reg-after-a-b (reg-file,
                                                                     real-mem),
                                                    c-flag,
                                                    v-flag),
```

```
alu-z-flag (v-nth (nth (0, reg-file),
                       real-mem),
                  a-reg-after-a-b (reg-file,
                                   real-mem),
                  b-reg-after-a-b (reg-file,
                                   real-mem),
                  c-flag,
                  z-flag),
alu-n-flag (v-nth (nth (0, reg-file),
                       real-mem),
                  a-reg-after-a-b (reg-file,
                                   real-mem),
                  b-reg-after-a-b (reg-file,
                                   real-mem),
                  c-flag,
                  n-flag),
a-reg-after-a-b (reg-file, real-mem),
b-reg-after-a-b (reg-file, real-mem),
v-nth (nth (0, reg-file), real-mem),
trunc (DEFAULT-VISUAL-MEM-VALUE, 16),
real-mem-thru-post-inc (v-nth (nth (0,
                                    reg-file),
                               real-mem),
                        c-flag,
                        v-flag,
                        z-flag,
                        n-flag,
                        addr-out-after-a-b (reg-file,
                                            real-mem),
                        real-mem,
                        a-reg-after-a-b (reg-file,
                                         real-mem),
                        b-reg-after-a-b (reg-file,
                                         real-mem)),
watch-dog (f,
           f,
           f,
           bv (bv-alu-cv (a-reg-after-a-b (reg-file,
                                           real-mem),
                          b-reg-after-a-b (reg-file,
                                           real-mem),
                          c-flag,
                          bv-alu-op-code (v-nth (nth (0,
                                                      reg-file),
```

real-mem)))),
addr-out-after-a-b (*reg-file*,
real-mem)),
cdr (*lst*)) **endif**

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                             ;;
;;          Abstraction for looking at certian registers in BIG-MACHINE.       ;;
;;          Specifically, the definition below allows us to view the registers ;;
;;          REG-FILE, REAL-MEM, C-FLAG, V-FLAG, Z-FLAG, and N-FLAG, which      ;;
;;          are the registers used in SOFT machine.                            ;;
;;                                                                             ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:
abstract (*m*)
= list (cadddddddr (*m*),
caddddddddddddddddddr (*m*),
caddddddddddr (*m*),
cadddddddddddr (*m*),
caddddddddddddr (*m*),
cadddddddddddddr (*m*))

```
; n-flag
```

THEOREM: abstract-list
(abstract (list (*mar*,
read,
write,
dtack,
reset,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,

visual-mem,
real-mem,
watch-dog))
= list (*reg-file*, *real-mem*, *c-flag*, *v-flag*, *z-flag*, *n-flag*))
= **t**

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;;         The following lemmas ensure SOFT machine does not change the       ;;
;;         size of any registers.                                             ;;
;;                                                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: ramp-reg-file-after-oprd-a-pre-decrement
((*m* = MACHINE-SIZE)
∧ ramp (*reg-file*, MACHINE-SIZE, 8)
∧ ramp (*real-mem*, MACHINE-SIZE, exp (2, MACHINE-SIZE)))
→ (every-member-sizep (reg-file-after-oprd-a-pre-decrement (*reg-file*, *real-mem*), *m*)
∧ (length (reg-file-after-oprd-a-pre-decrement (*reg-file*, *real-mem*)) = 8))

THEOREM: ramp-reg-file-after-oprd-b-pre-decrement
((*m* = MACHINE-SIZE)
∧ ramp (*reg-file*, MACHINE-SIZE, 8)
∧ ramp (*real-mem*, MACHINE-SIZE, exp (2, MACHINE-SIZE)))
→ (every-member-sizep (reg-file-after-oprd-b-pre-decrement (*reg-file*, *real-mem*), *m*)
∧ (length (reg-file-after-oprd-b-pre-decrement (*reg-file*, *real-mem*)) = 8))

THEOREM: size-bv-alu-cv-results
size (bv (bv-alu-cv-results (*reg-file*, *real-mem*, *c-flag*)))
= size (a-value-for-alu-after-oprd-a-pre-decrement (*reg-file*, *real-mem*))

THEOREM: ramp-reg-file-after-oprd-b-post-increment
((*m* = MACHINE-SIZE)
∧ ramp (*reg-file*, MACHINE-SIZE, 8)
∧ ramp (*real-mem*, MACHINE-SIZE, exp (2, MACHINE-SIZE)))
→ ramp (reg-file-after-oprd-b-post-increment (*reg-file*,
real-mem,

c-flag,
v-flag,
z-flag,
n-flag),
m,
8)

THEOREM: sizep-b-value-for-alu-after-oprd-b-pre-decrement
(ramp (*reg-file*, MACHINE-SIZE, 8)
∧ ramp (*real-mem*, MACHINE-SIZE, exp (2, MACHINE-SIZE)))
→ (size (b-value-for-alu-after-oprd-b-pre-decrement (*reg-file*, *real-mem*))
= MACHINE-SIZE)

THEOREM: ramp-real-mem-after-alu-write
((m = MACHINE-SIZE)
∧ (x = exp (2, MACHINE-SIZE))
∧ ramp (*reg-file*, MACHINE-SIZE, 8)
∧ ramp (*real-mem*, MACHINE-SIZE, exp (2, MACHINE-SIZE)))
→ ramp (real-mem-after-alu-write (*reg-file*,
real-mem,
c-flag,
v-flag,
z-flag,
n-flag),
m,
x)

THEOREM: size-nth
(ramp (*lst*, 16, n) ∧ ($i < n$)) → (size (nth (i, *lst*)) = 16)

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                                ;;
;;          The proof of equivalence between the hardware definition of          ;;
;;          FM8501 and the software definition of FM8501.  This lemma proves     ;;
;;          that BIG-MACHINE executes instructions in the same way as the        ;;
;;          interpreter SOFT.                                                     ;;
;;                                                                                ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: big-machine-is-soft-machine
((*mar1* = nat-to-bv (1, NXSZ))
∧ standard-hyps (*mar1*,
f,

f,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem))

→ (abstract (big-machine (*mar1*,
f,
f,
f,
f,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
watch-dog,
oracle (*reg-file*,
real-mem,
c-flag,
v-flag,
z-flag,
n-flag,
lst)))

= soft (*reg-file*, *real-mem*, *c-flag*, *v-flag*, *z-flag*, *n-flag*, *lst*))

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                              ;;
;;          The following events are concerning with the programmer's view     ;;
;;          of FM8501 during a reset operation.                                 ;;
;;                                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:
oracle-reset (*reg-file*, *real-mem*, *c-flag*, *v-flag*, *z-flag*, *n-flag*, *lst*)
= **if** *lst* ≃ **nil** **then nil**
else append (state-0-to-0-wait-to-1-oracle (car (car (*lst*))),
oracle (update-nth (**t**,
0,
reg-file,
nat-to-bv (0, MACHINE-SIZE)),
real-mem,
c-flag,
v-flag,
z-flag,
n-flag,
cdr (*lst*))) **endif**

THEOREM: ramp-reg-file-after-reset
(ramp (*reg-file*, MACHINE-SIZE, 8) ∧ (m = MACHINE-SIZE))
→ ramp (update-nth (**t**, 0, *reg-file*, nat-to-bv (0, m)), m, 8)

THEOREM: sizep-trunc-at-machine-size
(m = MACHINE-SIZE) → sizep (trunc (x, m), m)

```
;  Proof that reseting FM8501 sets register 0 to zero.
```

THEOREM: soft-reset-works
((*mar0* = nat-to-bv (0, NXSZ))
∧ standard-hyps (*mar0*,
f,
f,
dtack,
t,
no-store,
data-out,
reg-file,
addr-out,

c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem))
→ (abstract (big-machine (*mar0*,
f,
f,
dtack,
t,
no-store,
data-out,
reg-file,
addr-out,
c-flag,
v-flag,
z-flag,
n-flag,
a-reg,
b-reg,
i-reg,
visual-mem,
real-mem,
memory-watch-dog-history,
oracle-reset (*reg-file*,
real-mem,
c-flag,
v-flag,
z-flag,
n-flag,
lst)))
= soft-reset (*reg-file*, *real-mem*, *c-flag*, *v-flag*, *z-flag*, *n-flag*, *lst*))

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;;          Flag interpretation lemmas                                        ;;
;;                                                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

EVENT: Disable bv-alu-cv-results.

THEOREM: bit-vector-interpretation-of-z-flag
(bv-to-nat (bv (bv-alu-cv-results (*reg-file*, *real-mem*, *c-flag*))) ≃ 0)
= v-zerop (bv (bv-alu-cv-results (*reg-file*, *real-mem*, *c-flag*)))

THEOREM: integer-interpretation-of-z-flag
(bv-to-nat (bv (bv-alu-cv-results (*reg-file*, *real-mem*, *c-flag*))) ≃ 0)
= (0 = bv-to-tc (bv (bv-alu-cv-results (*reg-file*, *real-mem*, *c-flag*))))

THEOREM: bit-vector-interpretation-of-n-flag
negativep (bv-to-tc (bv (bv-alu-cv-results (*reg-file*, *real-mem*, *c-flag*))))
= bitn (bv (bv-alu-cv-results (*reg-file*, *real-mem*, *c-flag*)),
size (bv (bv-alu-cv-results (*reg-file*, *real-mem*, *c-flag*))))

Index