#|

|#

```
; The Correctness of a Piton Big Number Addition Program
; J Strother Moore

; This file corresponds to Chapter 3 of CLI Tech Report 22,
; "Piton:  A Verified Assembly Level Language".
```

EVENT: Start with the library "piton" using the compiled version.

EVENT: Enable plus-0.

EVENT: Enable commutativity-of-plus.

EVENT: Enable commutativity2-of-plus.

EVENT: Enable associativity-of-plus.

THEOREM: plus-add1-1

$((1 + x) + y) = (1 + (x + y))$

THEOREM: plus-add1-2
$(x + (1 + y)) = (1 + (x + y))$

EVENT: Enable equal-plus-0.

EVENT: Enable times-0.

EVENT: Enable times-add1.

EVENT: Enable times-distributes-over-plus.

EVENT: Enable commutativity-of-times.

EVENT: Enable commutativity2-of-times.

EVENT: Enable associativity-of-times.

EVENT: Enable equal-times-0.

EVENT: Enable difference-elim.

THEOREM: difference-elim-rewrite1
$((x \in \mathbf{N}) \wedge (x \not< y)) \rightarrow ((y + (x - y)) = x)$

EVENT: Enable lessp-remainder.

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

EVENT: Enable remainder-quotient-elim.

EVENT: Enable plus-remainder-times-quotient.

EVENT: Enable lessp-quotient.

THEOREM: lessp-quotient-linear
$((x \not\simeq \mathsf{0}) \wedge (y \neq \mathsf{1})) \rightarrow ((x \div y) < x)$

EVENT: Enable not-lessp-quotient.

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

DEFINITION:
nat->bign $(n,\ base,\ size)$
$=$ **if** $size \simeq \mathtt{0}$ **then nil**
else cons (tag (`'nat`, n **mod** $base$),
nat->bign $(n \div base,\ base,\ size - 1)$) **endif**

DEFINITION:
bign->nat $(a,\ base)$
$=$ **if** $a \simeq$ **nil then** `0`
else untag (car (a)) $+$ ($base$ $*$ bign->nat (cdr (a), $base$)) **endif**

DEFINITION:
bignp $(a,\ base)$
$=$ **if** $a \simeq$ **nil then** $a =$ **nil**
else listp (car (a))
$\land$ (type (car (a)) $=$ `'nat`)
$\land$ (untag (car (a)) $\in \mathbf{N}$)
$\land$ (untag (car (a)) $<$ $base$)
$\land$ (cddr (car (a)) $=$ **nil**)
$\land$ bignp (cdr (a), $base$) **endif**

THEOREM: bignp-nat->bign
$(base \not\simeq \mathtt{0}) \rightarrow$ bignp (nat->bign $(n,\ base,\ size)$, $base$)

THEOREM: length-nat->bign
$(base \not\simeq \mathtt{0}) \rightarrow$ (length (nat->bign $(n,\ base,\ size)$) $=$ fix $(size)$)

THEOREM: lessp-quotient-times
$(n < (b * e)) \rightarrow (((n \div b) < e) = \mathbf{t})$

THEOREM: bign->nat->bign
$((n \in \mathbf{N}) \land (\mathtt{1} < base) \land (n < \exp(base,\ size)))$
$\rightarrow$ (bign->nat (nat->bign $(n,\ base,\ size)$, $base$) $= n$)

DEFINITION:
tv->nat (c)
$=$ **if** c **then** `1`
else `0` **endif**

DEFINITION:
big-add-array (a, b, c, $base$)
= **if** $a \simeq$ **nil** **then nil**
else cons (tag (`'nat`,
(untag (car (a)) + untag (car (b)) + tv->nat (c))
mod $base$),
big-add-array (cdr (a),
cdr (b),
(untag (car (a))
+ untag (car (b))
+ tv->nat (c))
$\not<$ $base$,
$base$)) **endif**

DEFINITION:
big-add-carry-out (a, b, c, $base$)
= **if** $a \simeq$ **nil** **then** bool (c)
else big-add-carry-out (cdr (a),
cdr (b),
(untag (car (a))
+ untag (car (b))
+ tv->nat (c))
$\not<$ $base$,
$base$) **endif**

EVENT: Enable difference-x-x.

THEOREM: remainder-carryout1
$((i < base) \land (j < base) \land ((1 + (i + j)) \not< base))$
$\rightarrow \quad (((1 + (i + j)) \textbf{ mod } base) = ((1 + (i + j)) - base))$

THEOREM: remainder-carryout0
$((i < base) \land (j < base) \land ((i + j) \not< base))$
$\rightarrow \quad (((i + j) \textbf{ mod } base) = ((i + j) - base))$

THEOREM: interpretation-of-big-add-results
(bignp (a, $base$)
$\land$ bignp (b, $base$)
$\land$ (length (a) = length (b))
$\land$ ($base \in \mathbf{N}$)
$\land$ ($1 < base$))
$\rightarrow$ (bign->nat (append (big-add-array (a, b, c, $base$),
list (tag (`'nat`,
bool-to-nat (untag (big-add-carry-out (a,

b,
c,
$base$)))))),
$base$)
= (bign->nat (a, $base$) + bign->nat (b, $base$) + tv->nat (c)))

```
; The above theorem is just about all of the mathematical content of
; this exercise.  It remains relate the functions big-add-array and big-add-carry-out
; to a Piton program.
```

EVENT: Disable tv->nat.

THEOREM: equal-lessp-n-1
$(n < 1) = (n \simeq 0)$

DEFINITION:
big-add-array-loop (i, $a\text{-}array$, $b\text{-}array$, n, c, $base$)
= **if** $(n - 1) \simeq 0$
then put (tag ('nat,
(untag (get (i, $a\text{-}array$))
+ untag (get (i, $b\text{-}array$))
+ tv->nat (c))
mod $base$),
i,
$a\text{-}array$)
else big-add-array-loop (1 + i,
put (tag ('nat,
(untag (get (i, $a\text{-}array$))
+ untag (get (i, $b\text{-}array$))
+ tv->nat (c))
mod $base$),
i,
$a\text{-}array$),
$b\text{-}array$,
$n - 1$,
(untag (get (i, $a\text{-}array$))
+ untag (get (i, $b\text{-}array$))
+ tv->nat (c))
$\not<$ $base$,
$base$) **endif**

Definition:
big-add-carry-out-loop $(i,\ a\text{-}array,\ b\text{-}array,\ n,\ c,\ base)$

$$
\begin{array}{l}
= \ \ \textbf{if}\ (n - 1) \simeq \texttt{0} \\
\quad \textbf{then}\ \text{bool}((\text{untag}(\text{get}(i,\ a\text{-}array)) \\
\qquad\qquad + \ \text{untag}(\text{get}(i,\ b\text{-}array)) \\
\qquad\qquad + \ \text{tv->nat}(c)) \\
\qquad\qquad \not< \ base) \\
\quad \textbf{else}\ \text{big-add-carry-out-loop}\,(1 + i, \\
\qquad\qquad \text{put}\,(\text{tag}\,(\texttt{'nat}, \\
\qquad\qquad\qquad (\text{untag}(\text{get}(i,\ a\text{-}array)) \\
\qquad\qquad\qquad + \ \text{untag}(\text{get}(i,\ b\text{-}array)) \\
\qquad\qquad\qquad + \ \text{tv->nat}(c)) \\
\qquad\qquad\qquad \textbf{mod}\ \ base), \\
\qquad\qquad\quad i, \\
\qquad\qquad\quad a\text{-}array), \\
\qquad\qquad b\text{-}array, \\
\qquad\qquad n - 1, \\
\qquad\qquad (\text{untag}(\text{get}(i,\ a\text{-}array)) \\
\qquad\qquad + \ \text{untag}(\text{get}(i,\ b\text{-}array)) \\
\qquad\qquad + \ \text{tv->nat}(c)) \\
\qquad\qquad \not< \ base, \\
\qquad\qquad base)\ \textbf{endif}
\end{array}
$$

Definition:
nth-cdr $(i,\ a)$

$$
\begin{array}{l}
= \ \ \textbf{if}\ i \simeq \texttt{0}\ \ \textbf{then}\ a \\
\quad \textbf{else}\ \text{nth-cdr}\,(i - 1,\ \text{cdr}\,(a))\ \textbf{endif}
\end{array}
$$

Theorem: get-is-car-nth-cdr
get $(i,\ a) =$ car (nth-cdr $(i,\ a)$)

Event: Enable length-put.

Theorem: nth-cdr-add1
nth-cdr $(1 + i,\ x) =$ cdr (nth-cdr $(i,\ x)$)

Theorem: nth-cdr-put-at
$(i <$ length $(a))$
$\rightarrow$ (nth-cdr (i, put (val, i, a)) = cons (val, cdr (nth-cdr (i, a))))

Definition:
hint1 $(i,\ a,\ b,\ c,\ base)$

$$
\begin{array}{l}
= \ \ \textbf{if}\ i < \text{length}\,(a) \\
\quad \textbf{then}\ \text{hint1}\,(1 + i,
\end{array}
$$

$$
\begin{array}{l}
\quad\quad\quad\quad \text{put}\,(\text{tag}\,(\texttt{'nat}, \\
\quad\quad\quad\quad\quad\quad\quad\quad (\text{untag}\,(\text{get}\,(i,\ a)) \\
\quad\quad\quad\quad\quad\quad\quad\quad\ \ +\quad \text{untag}\,(\text{get}\,(i,\ b)) \\
\quad\quad\quad\quad\quad\quad\quad\quad\ \ +\quad \text{tv->nat}\,(c)) \\
\quad\quad\quad\quad\quad\quad\quad\quad \textbf{mod}\quad \mathit{base}), \\
\quad\quad\quad\quad\quad\quad i, \\
\quad\quad\quad\quad\quad\quad a), \\
\quad\quad\quad\quad b, \\
\quad\quad\quad\quad (\text{untag}\,(\text{get}\,(i,\ a)) + \text{untag}\,(\text{get}\,(i,\ b)) + \text{tv->nat}\,(c)) \\
\quad\quad\quad\quad \not< \quad \mathit{base}, \\
\quad\quad\quad\quad \mathit{base}) \\
\quad \textbf{else t endif}
\end{array}
$$

Theorem: listp-nth-cdr

$\text{listp}\,(\text{nth-cdr}\,(i,\ a)) = (i < \text{length}\,(a))$

Theorem: nth-cdr-put-before

$((j < i) \wedge (j < \text{length}\,(a)))$

$\rightarrow\quad (\text{nth-cdr}\,(i,\ \text{put}\,(\mathit{val},\ j,\ a)) = \text{nth-cdr}\,(i,\ a))$

Theorem: difference-equal-1

$((i < \text{length}\,(a)) \wedge (i \not< (\text{length}\,(a) - 1)))$

$\rightarrow\quad ((\text{length}\,(a) - i) = \texttt{1})$

Theorem: big-add-array-loop-on-list-of-length-1

$$
\begin{array}{l}
\text{big-add-array-loop}\,(i,\ a,\ b,\ \texttt{1},\ c,\ \mathit{base}) \\
= \quad \text{put}\,(\text{tag}\,(\texttt{'nat}, \\
\quad\quad\quad\quad\quad\quad (\text{untag}\,(\text{get}\,(i,\ a)) + \text{untag}\,(\text{get}\,(i,\ b)) + \text{tv->nat}\,(c)) \\
\quad\quad\quad\quad\quad\quad \textbf{mod}\quad \mathit{base}), \\
\quad\quad\quad\quad i, \\
\quad\quad\quad\quad a)
\end{array}
$$

Event: Enable equal-length-0.

Theorem: equal-length-1

$(\text{length}\,(a) = \texttt{1}) = (\text{listp}\,(a) \wedge (\text{cdr}\,(a) \simeq \textbf{nil}))$

Theorem: nlistp-cdr-nth-cdr

$\text{listp}\,(\text{cdr}\,(\text{nth-cdr}\,(i,\ a))) = (i < (\text{length}\,(a) - 1))$

Theorem: big-add-array-on-list-of-length-1

$$
\begin{array}{l}
(\text{listp}\,(a) \wedge (\text{cdr}\,(a) \simeq \textbf{nil})) \\
\rightarrow\quad (\text{big-add-array}\,(a,\ b,\ c,\ \mathit{base}) \\
\quad\quad\quad = \quad \text{list}\,(\text{tag}\,(\texttt{'nat}, \\
\quad\quad\quad\quad\quad\quad\quad\quad (\text{untag}\,(\text{car}\,(a)) + \text{untag}\,(\text{car}\,(b)) + \text{tv->nat}\,(c)) \\
\quad\quad\quad\quad\quad\quad\quad\quad \textbf{mod}\quad \mathit{base})))
\end{array}
$$

```
; The following lemma is intended only for use in the inductive step
; of mathematical-big-add-v-big-add-array-loop-generalized, where we
; replace i by (add1 i) and need to transform the lhs difference to
; the rhs sub1.
```

THEOREM: difference-add1-new
$(\text{length}\,(a) - (1 + i)) = ((\text{length}\,(a) - i) - 1)$

THEOREM: length-big-add-array-loop
$((i < \text{length}\,(a)) \wedge (\text{length}\,(a) \not< (i + n)))$
$\rightarrow \quad (\text{length}\,(\text{big-add-array-loop}\,(i,\ a,\ b,\ n,\ c,\ \mathit{base})) = \text{length}\,(a))$

THEOREM: car-nth-cdr-put
$((i \in \mathbf{N}) \wedge (j \in \mathbf{N}))$
$\rightarrow \quad (\text{car}\,(\text{nth-cdr}\,(i,\ \text{put}\,(\mathit{val},\ j,\ a)))$
$\quad\quad = \quad \textbf{if}\ i = j\ \textbf{then}\ \mathit{val}$
$\quad\quad\quad\quad \textbf{else}\ \text{car}\,(\text{nth-cdr}\,(i,\ a))\ \textbf{endif})$

THEOREM: car-nth-cdr-big-add-array-loop
$((k \in \mathbf{N}) \wedge (i \in \mathbf{N}) \wedge (k < i))$
$\rightarrow \quad (\text{car}\,(\text{nth-cdr}\,(k,\ \text{big-add-array-loop}\,(i,\ a,\ b,\ n,\ c,\ \mathit{base})))$
$\quad\quad = \quad \text{car}\,(\text{nth-cdr}\,(k,\ a)))$

THEOREM: equal-nth-cdr-nil
$\text{bignp}\,(a,\ \mathit{base}) \rightarrow ((\textbf{nil} = \text{nth-cdr}\,(i,\ a)) = (\text{fix}\,(i) = \text{length}\,(a)))$

THEOREM: equal-cddr-nth-cdr-nil
$\text{bignp}\,(a,\ \mathit{base})$
$\rightarrow \quad ((\textbf{nil} = \text{cddr}\,(\text{nth-cdr}\,(i,\ a))) = ((1 + (1 + i)) = \text{length}\,(a)))$

THEOREM: bignp-put
$((i \in \mathbf{N})$
$\wedge \quad (i < \text{length}\,(a))$
$\wedge \quad \text{bignp}\,(a,\ \mathit{base})$
$\wedge \quad (\mathit{val} \in \mathbf{N})$
$\wedge \quad (\mathit{val} < \mathit{base}))$
$\rightarrow \quad \text{bignp}\,(\text{put}\,(\text{list}\,(\texttt{'nat},\ \mathit{val}),\ i,\ a),\ \mathit{base})$

THEOREM: cons-car-cdr-hack
$(\text{cons}\,(a,\ \text{cdr}\,(x)) = x) = (\text{listp}\,(x) \wedge (a = \text{car}\,(x)))$

THEOREM: big-add-array-is-big-add-array-loop-generalized
$((i \in \mathbf{N})$
$\wedge \quad (i < \text{length}\,(a))$
$\wedge \quad \text{bignp}\,(a,\ \mathit{base})$

$\wedge \quad (base \in \mathbf{N})$
$\wedge \quad (\mathtt{1} < base))$
$\rightarrow \quad$ (big-add-array (nth-cdr $(i,\ a)$, nth-cdr $(i,\ b)$, $c,\ base$)
$\quad = \quad$ nth-cdr (i, big-add-array-loop ($i,\ a,\ b$, length $(a) - i,\ c,\ base$)))

THEOREM: big-add-array-is-big-add-array-loop
(listp (a) $\wedge$ bignp $(a,\ base)$ $\wedge$ $(base \in \mathbf{N})$ $\wedge$ $(\mathtt{1} < base)$)
$\rightarrow \quad$ (big-add-array $(a,\ b,\ c,\ base)$
$\quad = \quad$ big-add-array-loop ($\mathtt{0},\ a,\ b$, length (a), $c,\ base$))

THEOREM: big-add-carry-out-on-list-of-length-1
(listp (a) $\wedge$ (cdr $(a) \simeq$ **nil**))
$\rightarrow \quad$ (big-add-carry-out $(a,\ b,\ c,\ base)$
$\quad = \quad$ bool ((untag (car (a)) + untag (car (b)) + tv->nat (c))
$\qquad\qquad \not< \quad base$))

THEOREM: big-add-carry-out-is-big-add-carry-out-loop-generalized
$((i \in \mathbf{N}) \wedge (i <$ length $(a)))$
$\rightarrow \quad$ (big-add-carry-out (nth-cdr $(i,\ a)$, nth-cdr $(i,\ b)$, $c,\ base$)
$\quad = \quad$ big-add-carry-out-loop ($i,\ a,\ b$, length $(a) - i,\ c,\ base$))

THEOREM: big-add-carry-out-is-big-add-carry-out-loop
listp (a)
$\rightarrow \quad$ (big-add-carry-out $(a,\ b,\ c,\ base)$
$\quad = \quad$ big-add-carry-out-loop ($\mathtt{0},\ a,\ b$, length (a), $c,\ base$))

DEFINITION:
BIG-ADD-PROGRAM
```
=  '(big-add
     (a b n)
     nil
     (push-constant (bool f))
     (push-local a)
     (dl loop nil (fetch))
     (push-local b)
     (fetch)
     (add-nat-with-carry)
     (push-local a)
     (deposit)
     (push-local n)
     (sub1-nat)
     (set-local n)
     (test-nat-and-jump zero done)
     (push-local b)
     (push-constant (nat 1))
```

```
(add-addr)
(pop-local b)
(push-local a)
(push-constant (nat 1))
(add-addr)
(set-local a)
(jump loop)
(dl done nil (ret)))
```

DEFINITION:
big-add-loop-clock (n)
$=$ **if** $n \simeq \mathtt{0}$ **then** `0`
elseif $n = \mathtt{1}$ **then** `11`
else `19` $+$ big-add-loop-clock $(n - 1)$ **endif**

```
; The following clock function includes one tick for the CALL.  That is,
; BIG-ADD-CLOCK is the amount of time it takes to execute a CALL of BIG-ADD,
; not just the body of BIG-ADD.
```

DEFINITION: big-add-clock $(n) = (\mathtt{3} +$ big-add-loop-clock $(n))$

```
; The concept of value in the defs.events is at odds with the
; use of the term in the report.
```

DEFINITION: array $(name,\ segment) =$ cdr (assoc $(name,\ segment)$)

DEFINITION:
put-array $(a,\ name,\ segment) =$ put-assoc $(a,\ name,\ segment)$

EVENT: Disable booleanp.

DEFINITION: tv $(b) = (b = \mathtt{'t})$

EVENT: Enable assoc-put-assoc-2.

EVENT: Disable get-is-car-nth-cdr.

THEOREM: p-step1-opener
p-step1 (cons $(opcode,\ operands),\ p)$
$=$ **if** p-ins-okp (cons $(opcode,\ operands),\ p)$
then p-ins-step (cons $(opcode,\ operands),\ p)$
else p-halt $(p,$ x-y-error-msg $(\mathtt{'p},\ opcode))$ **endif**

EVENT: Disable p-step1.

THEOREM: p-opener

$$
\begin{aligned}
&(\mathrm{p}(s,\ \mathtt{0}) = s)\\
\wedge\quad &(\mathrm{p}(\text{p-state}(pc,\ ctrl,\ temp,\ prog,\ data,\ max\text{-}ctrl,\ max\text{-}temp,\ word\text{-}size,\ psw),\\
&\quad 1 + n)\\
&\quad = \ \mathrm{p}(\text{p-step}(\text{p-state}(pc,\ ctrl,\ temp,\ prog,\ data,\ max\text{-}ctrl,\ max\text{-}temp,\ word\text{-}size,\ psw)),\\
&\qquad n))
\end{aligned}
$$

EVENT: Disable p.

THEOREM: bignp-implies-p-objectp-get

$$
\begin{aligned}
&(\text{bignp}(a,\ \exp(\mathtt{2},\ word\text{-}size)) \wedge (i < \text{length}(a)))\\
\rightarrow\quad &(\text{listp}(\text{get}(i,\ a))\\
&\wedge\ (\text{cddr}(\text{get}(i,\ a)) = \mathbf{nil})\\
&\wedge\ (\text{car}(\text{get}(i,\ a)) = \mathtt{'nat})\\
&\wedge\ (\text{cadr}(\text{get}(i,\ a)) \in \mathbf{N})\\
&\wedge\ (\text{cadr}(\text{get}(i,\ a)) < \exp(\mathtt{2},\ word\text{-}size)))
\end{aligned}
$$

THEOREM: booleanp-not-f

$(\text{booleanp}(c) \wedge (c \neq \mathtt{'f})) \rightarrow ((c = \mathtt{'t}) = \mathtt{t})$

THEOREM: lessp-1-base

$(word\text{-}size \not\simeq \mathtt{0}) \rightarrow (\mathtt{1} < \exp(\mathtt{2},\ word\text{-}size))$

THEOREM: assoc-put-assoc-2-new

$(a \neq b) \rightarrow (\text{assoc}(a,\ \text{put-assoc}(val,\ b,\ segment)) = \text{assoc}(a,\ segment))$

EVENT: Enable definedp-put-assoc.

```
; ??? I am here.  The db is here.  The comment above is wrong.  Since then
; I have changed the theorem by eliminating the hyp (equal p (p-state ...)).
; I see no alternative to simply starting the proof of the thm below over again.
; That is what I have done.  The comment above remains just as a reminder of where
; I was.  It should be deleted when I start afresh.
```

THEOREM: booleanp-not-f-tv->nat
(booleanp $(c) \wedge (c \neq$ `'f`$)) \rightarrow$ (tv->nat $(c) =$ `1`)

DEFINITION:
big-add-loop-correct-hint (a, b, i, *data-segment*, c, *word-size*)
= **if** definedp (a, *data-segment*)
  $\wedge$ ($i <$ length (array (a, *data-segment*)))
 then if ((length (array (a, *data-segment*)) $-$ i) $-$ 1) $\simeq$ `0` **then t**
  else big-add-loop-correct-hint (a,
   b,
   $1 + i$,
   put-assoc (put (tag (`'nat`,
    (untag (get (i,
     array (a,
      data-segment)))
    + untag (get (i,
     array (b,
      data-segment)))
    + bool-to-nat (c))
    mod exp (`2`,
     word-size)),
   i,
   array (a,
    data-segment)),
   a,
   data-segment),
   if (untag (get (i,
    array (a,
     data-segment)))
    + untag (get (i,
     array (b,
      data-segment)))
    + bool-to-nat (c))
    < exp (`2`, *word-size*)
   then `'f`
   else `'t` **endif**,
   word-size) **endif**
 else t endif

EVENT: Enable put-assoc-put-assoc.

```
; I have proved 2 of the 3 cases of this.  I think the remaining case is
; trivial.  I will add it as an axiom and proceed.  The proof takes about
; 3 hours, judging from the two cases I did.
```

THEOREM: big-add-loop-correct-generalized

$$
\begin{array}{ll}
& ((\text{length}\,(\text{array}\,(a,\ \textit{data-segment})) < \exp\,(\texttt{2},\ \textit{word-size})) \\
\land & (\textit{word-size} \not\simeq \texttt{0}) \\
\land & \text{listp}\,(\textit{ctrl-stk}) \\
\land & \text{bignp}\,(\text{array}\,(a,\ \textit{data-segment}),\ \exp\,(\texttt{2},\ \textit{word-size})) \\
\land & \text{bignp}\,(\text{array}\,(b,\ \textit{data-segment}),\ \exp\,(\texttt{2},\ \textit{word-size})) \\
\land & (\textit{max-temp-stk-size} \not< (1 + (1 + (1 + \text{length}\,(\textit{temp-stk}))))) \\
\land & (\text{definition}\,(\texttt{'big-add},\ \textit{prog-segment}) = \text{BIG-ADD-PROGRAM}) \\
\land & \text{definedp}\,(a,\ \textit{data-segment}) \\
\land & \text{definedp}\,(b,\ \textit{data-segment}) \\
\land & (a \neq b) \\
\land & (i \in \mathbf{N}) \\
\land & (i < n) \\
\land & (n = \text{length}\,(\text{array}\,(a,\ \textit{data-segment}))) \\
\land & (n = \text{length}\,(\text{array}\,(b,\ \textit{data-segment}))) \\
\land & \text{booleanp}\,(c)) \\
\rightarrow & (\text{p}\,(\text{p-state}\,(\texttt{'(pc (big-add . 2))}, \\
& \qquad \text{cons}\,(\text{list}\,(\text{list}\,(\text{cons}\,(\texttt{'a},\ \text{list}\,(\texttt{'addr},\ \text{cons}\,(a,\ i))), \\
& \qquad\qquad \text{cons}\,(\texttt{'b},\ \text{list}\,(\texttt{'addr},\ \text{cons}\,(b,\ i))), \\
& \qquad\qquad \text{cons}\,(\texttt{'n},\ \text{list}\,(\texttt{'nat},\ n - i))), \\
& \qquad\qquad \textit{ret-pc}), \\
& \qquad\quad \textit{ctrl-stk}), \\
& \qquad \text{cons}\,(\text{list}\,(\texttt{'addr},\ \text{cons}\,(a,\ i)),\ \text{cons}\,(\text{list}\,(\texttt{'bool},\ c),\ \textit{temp-stk})), \\
& \qquad \textit{prog-segment}, \\
& \qquad \textit{data-segment}, \\
& \qquad \textit{max-ctrl-stk-size}, \\
& \qquad \textit{max-temp-stk-size}, \\
& \qquad \textit{word-size}, \\
& \qquad \texttt{'run}), \\
& \quad \text{big-add-loop-clock}\,(n - i)) \\
& = \ \text{p-state}\,(\textit{ret-pc}, \\
& \qquad \textit{ctrl-stk}, \\
& \qquad \text{cons}\,(\text{big-add-carry-out-loop}\,(i, \\
& \qquad\qquad \text{array}\,(a,\ \textit{data-segment}), \\
& \qquad\qquad \text{array}\,(b,\ \textit{data-segment}), \\
& \qquad\qquad n - i, \\
& \qquad\qquad \text{tv}\,(c), \\
& \qquad\qquad \exp\,(\texttt{2},\ \textit{word-size})), \\
& \qquad\quad \textit{temp-stk}), \\
& \qquad \textit{prog-segment}, \\
& \qquad \text{put-assoc}\,(\text{big-add-array-loop}\,(i, \\
& \qquad\qquad \text{array}\,(a,\ \textit{data-segment}),
\end{array}
$$

array (b, *data-segment*),
$n - i$,
tv (c),
exp (2, *word-size*)),
a,
data-segment),
max-ctrl-stk-size,
max-temp-stk-size,
word-size,
`'run`))

Theorem: big-add-loop-correct
((length (array (a, *data-segment*)) $<$ exp (2, *word-size*))
$\land$ (*word-size* $\not\simeq$ 0)
$\land$ listp (*ctrl-stk*)
$\land$ bignp (array (a, *data-segment*), exp (2, *word-size*))
$\land$ bignp (array (b, *data-segment*), exp (2, *word-size*))
$\land$ (*max-temp-stk-size* $\not<$ (1 + (1 + (1 + length (*temp-stk*)))))
$\land$ (definition (`'big-add`, *prog-segment*) = BIG-ADD-PROGRAM)
$\land$ definedp (a, *data-segment*)
$\land$ definedp (b, *data-segment*)
$\land$ ($a \neq b$)
$\land$ ($n \not\simeq 0$)
$\land$ (n = length (array (a, *data-segment*)))
$\land$ (n = length (array (b, *data-segment*)))
$\land$ booleanp (c))
$\rightarrow$ (p (p-state (`'(pc (big-add . 2))`,
cons (list (list (cons (`'a`, list (`'addr`, cons (a, 0))),
cons (`'b`, list (`'addr`, cons (b, 0))),
cons (`'n`, list (`'nat`, n))),
ret-pc),
ctrl-stk),
cons (list (`'addr`, cons (a, 0)),
cons (list (`'bool`, c), *temp-stk*)),
prog-segment,
data-segment,
max-ctrl-stk-size,
max-temp-stk-size,
word-size,
`'run`),
big-add-loop-clock (n))
= p-state (*ret-pc*,
ctrl-stk,
cons (big-add-carry-out-loop (0,

```
                                              array (a, data-segment),
                                              array (b, data-segment),
                                              n,
                                              tv (c),
                                              exp (2, word-size)),
                      temp-stk),
                 prog-segment,
                 put-assoc (big-add-array-loop (0,
                                               array (a, data-segment),
                                               array (b, data-segment),
                                               n,
                                               tv (c),
                                               exp (2, word-size)),
                           a,
                           data-segment),
                 max-ctrl-stk-size,
                 max-temp-stk-size,
                 word-size,
                 'run))
```

DEFINITION:
big-add-input-conditionp $(a,\ b,\ n,\ p0)$

```
=    (definedp (a, p-data-segment (p0))
      ∧  definedp (b, p-data-segment (p0))
      ∧  (a ≠ b)
      ∧  bignp (array (a, p-data-segment (p0)), exp (2, p-word-size (p0)))
      ∧  bignp (array (b, p-data-segment (p0)), exp (2, p-word-size (p0)))
      ∧  (n = length (array (a, p-data-segment (p0))))
      ∧  (n = length (array (b, p-data-segment (p0))))
      ∧  (p-max-ctrl-stk-size (p0)
            ≮  (5 + p-ctrl-stk-size (p-ctrl-stk (p0))))
      ∧  (p-max-temp-stk-size (p0) ≮ (3 + length (p-temp-stk (p0))))
      ∧  (n ≄ 0)
      ∧  (n < exp (2, p-word-size (p0)))
      ∧  listp (p-ctrl-stk (p0)))
```

THEOREM: not-zerop-wordsize-hack

$((n < \exp(2,\ word\text{-}size)) \land (n \not\simeq 0))$
$\rightarrow\ (((word\text{-}size \in \mathbf{N}) = \mathbf{t}) \land (word\text{-}size \neq 0))$

THEOREM: big-add-correct

```
 ((p0 = p-state (pc,
                 ctrl-stk,
                 append (list (tag ('nat, n),
                               tag ('addr, cons (b, 0)),
```

```
                    tag ('addr, cons (a, 0))),
                temp-stk),
        prog-segment,
        data-segment,
        max-ctrl-stk-size,
        max-temp-stk-size,
        word-size,
        'run))
 ∧  (p-current-instruction (p0) = '(call big-add))
 ∧  (definition ('big-add, prog-segment) = BIG-ADD-PROGRAM)
 ∧  big-add-input-conditionp (a, b, n, p0))
→   (p (p0, big-add-clock (n))
     =  p-state (add1-addr (pc),
                ctrl-stk,
                push (big-add-carry-out (array (a, data-segment),
                                         array (b, data-segment),
                                         f,
                                         exp (2, word-size)),
                      temp-stk),
                prog-segment,
                put-array (big-add-array (array (a, data-segment),
                                          array (b, data-segment),
                                          f,
                                          exp (2, word-size)),
                           a,
                           data-segment),
                max-ctrl-stk-size,
                max-temp-stk-size,
                word-size,
                'run))
```

```
; The above thm is not useful as a rewrite rule because p0 occurs
; in the lhs and pc, ctrl-stk, etc., are all free vars.  So to make it
; a rewrite rule we actually substitute (p-state pc ...) for p0 into the
; lhs.  In addition, the append in that initial state is going to be
; conses in applications, so we expand it in the lhs.  In addition,
; the tags are expanded to lists.
```

THEOREM: big-add-correct-rewrite-version

```
((p0 = p-state (pc,
        ctrl-stk,
        append (list (tag ('nat, n),
                      tag ('addr, cons (b, 0)),
```

```
                        tag ('addr, cons (a, 0))),
                     temp-stk),
              prog-segment,
              data-segment,
              max-ctrl-stk-size,
              max-temp-stk-size,
              word-size,
              'run))
 ∧   (p-current-instruction (p0) = '(call big-add))
 ∧   (definition ('big-add, prog-segment) = BIG-ADD-PROGRAM)
 ∧   big-add-input-conditionp (a, b, n, p0))
 →   (p (p-state (pc,
              ctrl-stk,
              cons (list ('nat, n),
                   cons (list ('addr, cons (b, 0)),
                        cons (list ('addr, cons (a, 0)), temp-stk))),
              prog-segment,
              data-segment,
              max-ctrl-stk-size,
              max-temp-stk-size,
              word-size,
              'run),
        big-add-clock (n))
     =  p-state (add1-addr (pc),
              ctrl-stk,
              push (big-add-carry-out (array (a, data-segment),
                                       array (b, data-segment),
                                       f,
                                       exp (2, word-size)),
                    temp-stk),
              prog-segment,
              put-array (big-add-array (array (a, data-segment),
                                        array (b, data-segment),
                                        f,
                                        exp (2, word-size)),
                         a,
                         data-segment),
              max-ctrl-stk-size,
              max-temp-stk-size,
              word-size,
              'run))
```

EVENT: Disable not-zerop-wordsize-hack.

```
;  That says all we need to say about BIG-ADD except for how we connect
; this thm to our FM8502 correctness result.  The question is: what has
; to be true about a state p in order for the hyp of

#|
(IMPLIES (AND (PROPER-P-STATEP P0)
              (P-LOADABLEP P0)
              (EQUAL (P-WORD-SIZE P0) 32)
              (EQUAL PN (P P0 N))
              (NOT (ERRORP (P-PSW PN)))
              (EQUAL TS (TYPE-SPECIFICATION (P-DATA-SEGMENT PN))))
         (EQUAL (P-DATA-SEGMENT PN)
                (DISPLAY-M-DATA-SEGMENT
                  (FM8502 (LOAD P0) (FM8502-CLOCK P0 N))
                  TS
                  (LINK-TABLES P0))))
|#

; to be true?

; Let us first identify a state that is suitable for running BIG-ADD.
```

DEFINITION:
MAIN-PROGRAM
```
=  '(main nil nil
     (push-constant (addr (bna . 0)))
     (push-constant (addr (bnb . 0)))
     (push-global n)
     (call big-add)
     (pop-global c)
     (ret))
```

DEFINITION:
system-initial-state (a, b)
= p-state (`'(pc (main . 0))`,
`'((nil (pc (main . 0))))`,
nil,
list (MAIN-PROGRAM, BIG-ADD-PROGRAM),
list (cons (`'bna`, a),
cons (`'bnb`, b),
cons (`'n`, list (tag (`'nat`, length (a)))),
cons (`'c`, list (tag (`'nat`, 0)))),
10,

```
8,
32,
'run)
```

```
; This state takes the following number of clock ticks:
```

DEFINITION:
system-initial-state-clock $(a,\ b) = (5 + \text{big-add-clock}\,(\text{length}\,(a)))$

```
; The condition under which big-add-initial-state produces a acceptable state for
; the abstract computation to work is
```

EVENT: Disable exp.

EVENT: Disable *1*exp.

DEFINITION:
system-initial-state-okp $(a,\ b)$
$$
\begin{aligned}
= \quad & (\text{bignp}\,(a,\ \text{exp}\,(2,\ 32)) \\
& \wedge \quad \text{bignp}\,(b,\ \text{exp}\,(2,\ 32)) \\
& \wedge \quad (\text{length}\,(a) = \text{length}\,(b)) \\
& \wedge \quad (\text{length}\,(a) \not\simeq 0) \\
& \wedge \quad (\text{length}\,(a) < \text{exp}\,(2,\ 32)))
\end{aligned}
$$

```
; We prove that system-initial-state-okp lives up to the above claim:
```

THEOREM: restructure-system-initial-state-clock
system-initial-state-clock $(a,\ b) = (3 + \text{big-add-clock}\,(\text{length}\,(a)) + 2)$

EVENT: Disable system-initial-state-clock.

EVENT: Disable plus-add1-2.

EVENT: Disable commutativity-of-plus.

THEOREM: p-plus
$\text{p}\,(p,\ i + j) = \text{p}\,(\text{p}\,(p,\ i),\ j)$

THEOREM: system-correct
system-initial-state-okp $(a,\ b)$

$\rightarrow$ (p (system-initial-state (a, b), system-initial-state-clock (a, b))
  = p-state (`'(pc (main . 5))`,
      `'((nil (pc (main . 0))))`,
      nil,
      list (MAIN-PROGRAM, BIG-ADD-PROGRAM),
      list (cons (`'bna`, big-add-array (a, b, **f**, exp (2, 32))),
        cons (`'bnb`, b),
        cons (`'n`, list (tag (`'nat`, length (a)))),
        cons (`'c`,
          list (big-add-carry-out (a, b, **f**, exp (2, 32))))),
      10,
      8,
      32,
      `'halt`))

```
; Furthermore, if system-initial-state-okp is true then the initial state is
; proper-p-state and its word size is 32, as required by our FM8502 correctness
; result.
```

THEOREM: bignp-implies-all-p-objectps
(bignp (a, *base*) $\wedge$ (*base* $=$ exp (2, p-word-size (p))))
$\rightarrow$ all-p-objectps (a, p)

THEOREM: initial-correctness-conditions
system-initial-state-okp (a, b)
$\rightarrow$ (proper-p-statep (system-initial-state (a, b))
  $\wedge$ (p-word-size (system-initial-state (a, b)) $=$ 32))

```
; In addition, the final state is not an error and its type-specification is
; easily computed from the input:
```

DEFINITION:
nat-lst (n)
$=$ **if** $n \simeq 0$ **then nil**
  else cons (`'nat`, nat-lst ($n - 1$)) **endif**

THEOREM: type-lst-big-add-array
type-lst (big-add-array (a, b, c, *base*)) $=$ nat-lst (length (a))

THEOREM: type-lst-bignp
bignp (a, *base*) $\rightarrow$ (type-lst (a) $=$ nat-lst (length (a)))

THEOREM: car-big-add-carry-out
car (big-add-carry-out (a, b, c, *base*)) $=$ `'bool`

THEOREM: final-correctness-conditions

$$\begin{array}{l}
\text{system-initial-state-okp}\,(a,\ b) \\
\rightarrow \quad ((\neg\ \text{errorp}\,(\text{p-psw}\,(\text{p}\,(\text{system-initial-state}\,(a,\ b), \\
\qquad\qquad\qquad\qquad \text{system-initial-state-clock}\,(a,\ b))))) \\
\qquad \wedge \quad (\text{type-specification}\,(\text{p-data-segment}\,(\text{p}\,(\text{system-initial-state}\,(a,\ b), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{system-initial-state-clock}\,(a, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad b)))) \\
\qquad\qquad = \quad \text{list}\,(\text{cons}\,(\texttt{'bna}, \text{nat-lst}\,(\text{length}\,(a))), \\
\qquad\qquad\qquad\quad \text{cons}\,(\texttt{'bnb}, \text{nat-lst}\,(\text{length}\,(a))), \\
\qquad\qquad\qquad\quad \text{cons}\,(\texttt{'n}, \text{list}\,(\texttt{'nat})), \\
\qquad\qquad\qquad\quad \text{cons}\,(\texttt{'c}, \text{list}\,(\texttt{'bool})))))
\end{array}$$

```
; This leaves only the question, is the system-initial-state loadable?  That all depends on
; how much room you take up with the data.  It is certainly loadable for some
; values of a and b and some load addresses.  Below we use the load address 0.
```

THEOREM: loadable-for-bigns-up-to-length-1000

$$\begin{array}{l}
(\text{system-initial-state-okp}\,(a,\ b) \wedge (\text{length}\,(a) < \texttt{1000})) \\
\rightarrow \quad \text{p-loadablep}\,(\text{system-initial-state}\,(a,\ b),\ \texttt{0})
\end{array}$$

```
; Oh, there is one more thing we might ask, which is whether there are
; any system-initial-state-okps.  There are.
```

THEOREM: example-is-system-initial-state-okp

$$\begin{array}{l}
\text{system-initial-state-okp}\,(\texttt{'((nat 246838082)} \\
\qquad\qquad\qquad\qquad\quad \texttt{(nat 3116233281)} \\
\qquad\qquad\qquad\qquad\quad \texttt{(nat 42632655)} \\
\qquad\qquad\qquad\qquad\quad \texttt{(nat 0))}, \\
\qquad\qquad\qquad\qquad \texttt{'((nat 3579363592)} \\
\qquad\qquad\qquad\qquad\quad \texttt{(nat 3979696680)} \\
\qquad\qquad\qquad\qquad\quad \texttt{(nat 7693250)} \\
\qquad\qquad\qquad\qquad\quad \texttt{(nat 0))})
\end{array}$$

THEOREM: type-specification-hack

$$\text{system-initial-state-okp}\,(a,\ b) \rightarrow (\text{type-lst}\,(b) = \text{nat-lst}\,(\text{length}\,(b)))$$

```
; We need to refer to the link tables for a state without knowing
; the actual data, just the size of the data.  So we define the
; function below which constructs the link tables from a generic
; state of the right size.  This will turn out to be EQUAL to the
; link table for any state of that size.
```

DEFINITION:
big-zero (n)
$=$ **if** $n \simeq$ `0` **then nil**
else cons (`'(nat 0)`, big-zero $(n-1)$) **endif**

```
; We link for load address 0.
```

DEFINITION:
system-link-tables $(n,\ m)$
$=$ link-tables (system-initial-state (big-zero (n), big-zero (m)), `0`)

THEOREM: length-big-zero
length (big-zero (n)) $=$ fix (n)

THEOREM: link-tables-length
link-tables (system-initial-state $(a,\ b)$, `0`)
$=$ system-link-tables (length (a), length (b))

THEOREM: system-initial-state-okp-length-hack
system-initial-state-okp $(a,\ b)$ $\rightarrow$ (length (b) $=$ length (a))

DEFINITION:
display-answers (*m-state*, n)
$=$ **let** *alist* **be** display-fm9001-data-segment (*m-state*,
list (cons (`'bna`,
nat-lst (n)),
cons (`'bnb`,
nat-lst (n)),
cons (`'n`, list (`'nat`)),
cons (`'c`,
list (`'bool`))),
system-link-tables $(n,\ n)$)
in
list (cdr (assoc (`'bna`, *alist*)), cadr (assoc (`'c`, *alist*))) **endlet**

THEOREM: type-big-add-carry-out
type (big-add-carry-out $(a,\ b,\ c,\ base)$) $=$ `'bool`

THEOREM: eliminate-piton
(system-initial-state-okp $(a,\ b)$ $\wedge$ (length (a) $<$ `1000`))
$\rightarrow$ (display-answers (fm9001 (load (system-initial-state $(a,\ b)$, **nil**, `0`),
fm9001-clock (system-initial-state $(a,\ b)$,
system-initial-state-clock $(a,\ b)$)),
length (a))
$=$ list (big-add-array $(a,\ b,$ **f**, exp (`2`, `32`)),
big-add-carry-out $(a,\ b,$ **f**, exp (`2`, `32`))))

THEOREM: clock-for-concrete-data

let a **be** `'((nat 246838082)`
`(nat 3116233281)`
`(nat 42632655)`
`(nat 0))`,
b **be** `'((nat 3579363592)`
`(nat 3979696680)`
`(nat 7693250)`
`(nat 0))`
in
fm9001-clock (system-initial-state $(a,\ b)$,
system-initial-state-clock $(a,\ b)$) **endlet**
$=$ `190`

Index