Floyd-Hoare Logic:

• A formal system for proving correctness
• A program operates on state - moving through code changes state
• Hoare logic follows state changes through triples:

\[ \{P\} \ C \ \{Q\} \]

where \( P \) is an assertion about the state before the execution of line \( C \) and \( Q \) is an assertion about the state after execution of \( C \).

• Examples:

\[ \{P\} \ \text{nop} \ \{P\} \]
\[ \{a \ == \ 1, \ a \ != \ b\} \ b := 2 \ \{a \ == \ 1, \ a \ != \ b\} \]
\[ \{??\} \ a := b+1 \ \{a > 1\} \]
Floyd-Hoare Logic:

- A formal system for proving correctness
- A program operates on state - moving through code changes state
- Hoare logic follows state changes through triples:

  \[
  \{P\} \ C \ \{Q\}
  \]

  where \(P\) is an assertion about the state before the execution of line \(C\) and \(Q\) is an assertion about the state after execution of \(C\).

- **Examples:**

  \[
  \{P\} \ \text{nop} \ \{P\}
  \]

  \[
  \{a \ == \ 1, \ a \ != \ b\} \ b \ := \ 2 \ \{a \ == \ 1, \ a \ != \ b\}
  \]

  \[
  \{??\} \ a \ := \ b+1 \ \{a \ > \ 1\}
  \]

  \[
  \{b \ == \ 20\} \ a \ := \ b+1 \ \{a \ > \ 1\}
  \]

  \[
  \{b \ > \ 10\} \ a \ := \ b+1 \ \{a \ > \ 1\}
  \]

  \[
  \{b \ > \ 0\} \ a \ := \ b+1 \ \{a \ > \ 1\}
  \]

  **Weakest precondition**
Proving Programs Correct

Floyd-Hoare Logic:

- A formal system for proving correctness
- A program operates on state - moving through code changes state
- Hoare logic follows state changes through triples:

\[ \{P\} \ C \ \{Q\} \]

where \( P \) is an assertion about the state before the execution of line \( C \) and \( Q \) is an assertion about the state after execution of \( C \).

- **Examples:**

\[ \{P\} \ \text{nop} \ \{P\} \]
\[ \{a \ == \ 1, \ a \ != \ b\} \ b := 2 \ \{a \ == \ 1, \ a \ != \ b\} \]
\[ \{b > 10\} \ a := b+1 \ \{??\} \]
Proving Programs Correct

Floyd-Hoare Logic:

- A formal system for proving correctness
- A program operates on state - moving through code changes state
- Hoare logic follows state changes through triples:
  
  \[ \{P\} \ C \ \{Q\} \]

  where \( P \) is an assertion about the state before the execution of line \( C \) and \( Q \) is an assertion about the state after execution of \( C \).

- **Examples:**

  \[ \{P\} \ \text{nop} \ \{P\} \]
  
  \[ \{a == 1, a != b\} \ b := 2 \ \{a == 1, a != b\} \]
  
  \[ \{b > 10\} \ a := b+1 \ \{??\} \]
  
  \[ \{b > 10\} \ a := b+1 \ \{a > 0\} \]
  
  \[ \{b > 10\} \ a := b+1 \ \{a > 11\} \quad \text{Strongest postcondition} \]
Proving Programs Correct

Floyd-Hoare Logic:

- **A loop invariant:**
  Used to prove loop properties. Assertions on state entering the loop and guaranteed to be true at every iteration of the loop.
  The invariant will be the postcondition for the loop on exit.

- **Example:**
  
  ```
  while (x < 10) x = x+1
  ```
Floyd-Hoare Logic:

- **A loop invariant:**
  
  Used to prove loop properties. Assertions on state entering the loop and guaranteed to be true at every iteration of the loop.
  
  The invariant will be the postcondition for the loop on exit.

- **Example:**
  
  ```plaintext
  while (x < 10) x = x+1
  
  Start with this:
  ```

  ```plaintext
  {x <= 10} while (x < 10) x = x+1 {??}
  ```
Floyd-Hoare Logic:

- **A loop invariant:**
  
  Used to prove loop properties. Assertions on state entering the loop and guaranteed to be true at every iteration of the loop. The invariant will be the postcondition for the loop on exit.

- **Example:**

  ```
  while (x < 10) x = x+1
  
  Start with this:
  
  \{x <= 10\} while (x < 10) x = x+1 \{??\}
  
  Move inside the test:
  
  \{x < 10 \land x <= 10\} x = x+1 \{x <= 10\}
  ```
Proving Programs Correct

Floyd-Hoare Logic:

- **A loop invariant:**
  
  Used to prove loop properties. Assertions on state entering the loop and guaranteed to be true at every iteration of the loop. The invariant will be the postcondition for the loop on exit.

- **Example:**

  
  while (x < 10) x = x+1
  
  Start with this:
  
  $$\{x \leq 10\} \text{ while } (x < 10) \ x = x+1 \ {??}\$$
  
  Move inside the test:
  
  $$\{x < 10 \land x \leq 10\} \ x = x+1 \ {x \leq 10}\$$
  
  Backing out:
  
  $$\{x < 10 \land x \leq 10\} \text{ while } (x<10) \ x=x+1 \ {\lnot(x < 10) \land x \leq 10}\$$
Floyd-Hoare Logic Rules:

- **Assignment:**
  Let $P[E/x]$ mean that in predicate $P$, expression $E$ is substituted for symbol $x$ where $x$ is a free variable.

  $\{P[E/x]\} \ x := E \ {P}$

- **Example:**
  In  $\{y == 10\} \ x := y+1 \ \{x == 11\}$
  $P = \{x == 11\}$
  $E/x = (y + 1)/x = \{y+1\}$
  $P[E/x] = \{y+1 == 11\} = \{y == 10\}$
Proving Programs Correct

Floyd-Hoare Logic Rules:

• Composition:

\[
\begin{align*}
\{P\} & \quad C_1 \quad \{Q\}, \quad \{Q\} \quad C_2 \quad \{R\} \\
\{P\} & \quad C_1; C_2 \quad \{R\}
\end{align*}
\]

• Example:

In \{y == 10\} \ x := y+1 \ \{x == 11\} \ \text{and} \\
\{x == 11\} \ z := x+1 \ \{z == 12\}

We can substitute \\
\{y == 10\} \ x := y+1; z := x+1 \ \{z == 12\}
Proving Programs Correct

Floyd-Hoare Logic Rules:

- **Condition:**

\[
\begin{align*}
\{A \land B\} & \quad C_1 \quad \{Q\}, \quad \{-A \land B\} \quad C_2 \quad \{Q\} \\
\{B\} \quad \text{if} \quad A \quad \text{then} \quad C_1 \quad \text{else} \quad C_2 \quad \{Q\}
\end{align*}
\]

- **Example:**

In \[\{a == T \land b == 6 \land c == 10\}\] \[x := b\]
\[
\{(x == 6 \land a == T) \lor (x == 10 \land a == F)\}
\]

and \[\{a == F \land b == 6 \land c == 10\}\] \[x := c\]
\[
\{(x == 6 \land a == T) \lor (x == 10 \land a == F)\}
\]

After applying the condition rule:
\[
\{(b == 6 \land c == 10)\}
\]
If \[a == T\] then \[x := b\]; else \[x := c\]
\[
\{(x == 6 \land a == T) \lor (x == 10 \land a == F)\}\]
Floyd-Hoare Logic Rules:

- **Consequence:**
  
  \[
  P' \rightarrow P, \quad \{P\} \ C \ \{Q\}, \quad Q' \rightarrow Q
  \]
  
  \[
  \frac{}{\{P'\} \ C \ \{Q'\}}
  \]

- **Example:**
  
  Starting with:
  
  \[
  \{P\} = \{x + 1 == 10\} \quad y := x + 1 \quad \{y == 10\} = \{Q\}
  \]
  
  and
  
  \[
  P' = (x == 9 \rightarrow x + 1 == 10),
  \]
  
  \[
  Q' = (x == 9 \land y == 10 \rightarrow y == 10),
  \]
  
  application of the consequence rule gives:
  
  \[
  \{x == 9\} \quad y := x + 1 \quad \{x == 9 \land y == 10\}
  \]
Proving Programs Correct

Floyd-Hoare Logic Rules:

- **While:**

\[
\{P \land A\} C \{P\} \\
\{P\} \text{ while } A \text{ do } C \{\neg A \land P\}
\]

- **Example:**

Starting with:

\[
\{x < 10 \land x \leq 10\} \ x = x + 1; \ \{x \leq 10\}
\]

After application of the while rule:

\[
\{x \leq 10\} \ \text{ while}(x < 10) \ x = x + 1; \ \{\neg(x < 10) \land x \leq 10\}
\]

After application of the consequence rule:

\[
\{x \leq 10\} \ \text{ while}(x < 10) \ x = x + 1; \ \{x == 10\}
\]
Weakest Precondition:

If $Q$ is a predicate on states and $C$ is a code fragment, then the weakest precondition for $C$ with respect to $Q$ is a predicate that is true for precisely those initial states in which $C$ must terminate and must produce a state satisfying $Q$.

Notation:

$\text{wp}(C, Q)$ - weakest precondition wrt $Q$, given code $C$.

Example:

$\{P\} \ a := b + 1 \ {a > 11}$

$P = \text{wp}(a := b + 1, \ a > 11) = b > 10$

Nomenclature:

$\text{wp}$ is called a predicate transformer
Proving Programs Correct

Weakest Precondition:

Example:

\( \{ Q(y + 3 \times z - 5) \} \ x := y + 3 \times z - 5 \ \{ Q(x) \} \)

Let \( v \) be the value assigned to \( x \).

If \( Q(x) \) is true after assignment so is \( Q(v) \) (before and after)

Then \( Q(y + 3 \times z - 5) \) is true initially

So, \( Q(x) \) holds after assignment iff \( Q(y + 3 \times z - 5) \) held

Usefulness:

Any predicate that implies \( \text{wp}(C,Q) \) also implies \( Q \)

Hence, if \( C \) is an entire program and \( Q \) specifies a property of \( C \) and if \( \text{wp}(C,Q) \) is known to be the weakest precondition from the initial state, ACL2 can be used to determine input values that cause \( Q \) to be true.
Weakest Precondition:

1. from initial state to state not satisfying Q
2. from initial state to some state satisfying Q
3. from initial state to states satisfying Q
Proving Programs Correct

Strongest Postcondition:

If \( P \) is a predicate on states and \( C \) is a code fragment, then the strongest postcondition for \( C \) with respect to \( P \) is a predicate that is implied by \( P \) when acted upon by \( C \).

Notation:

\[ \text{sp}(C, P) \] - strongest postcondition wrt \( P \), given code \( C \).

Example:

\( \{b > 10\} \ a := b + 1 \ \{Q\} \)

\[ Q = \text{sp}(a := b + 1, \ b > 10) = a > 11 \]

Nomenclature:

\( \text{sp} \) is also called a predicate transformer
Proving Programs Correct

Predicate Transformers:

- PT semantics are a reformulation of Floyd-Hoare logic
- Used to implement an effective process for transforming F-H logic to predicate logic (so ACL2 can operate on it).

**WP - nop:**

\[ wp(nop, Q) = Q \]

**WP - abort:**

\[ wp(abort, Q) = false \]

**WP - assignment:**

\[ wp(x := E, Q) = \forall z. z == E \rightarrow Q[z/x] \]
\[ wp(x := E, Q) = Q[E/x] \]

ex.: \[ wp(x := x + 1, x > 10) = x + 1 > 10 \iff x > 9 \]
Proving Programs Correct

WP - composition:
\[ \text{wp}(C_1; C_2, Q) = \text{wp}(C_1, \text{wp}(C_2, Q)) \]

ex.: \[ \text{wp}(x := x + 2; y := y - 2, (x + y == 0)) \]
\[ = \text{wp}(x := x + 2, \text{wp}(y := y - 2, (x + y == 0))) \]
\[ = \text{wp}(x := x + 2, (x + (y - 2) == 0)) \]
\[ = ((x + 2) + y - 2) == 0 \]
\[ = x + y == 0 \]

WP - condition:
\[ \text{wp}(\text{if } E \text{ then } C_1 \text{ else } C_2, Q) = (E \rightarrow \text{wp}(C_1, Q)) \land (\neg E \rightarrow \text{wp}(C_2, Q)) \]

ex.: \[ \text{wp}(\text{if } x > 2 \text{ then } y := 1 \text{ else } y := -1, (y > 0)) \]
\[ = ((x > 2) \rightarrow \text{wp}(y := 1, (y > 0))) \land \]
\[ (\neg(x > 2) \rightarrow \text{wp}(y := -1, (y > 0))) \]
\[ = ((x > 2) \rightarrow (1 > 0)) \land (\neg(x > 2) \rightarrow (1 > 0)) \]
\[ = ((x > 2) \rightarrow T) \land (\neg(x > 2) \rightarrow F)) \]
\[ = (x > 2) \land ((x > 2) \lor \text{F}) \]
\[ = (x > 2) \]
Proving Programs Correct

WP - while:

\[ \text{wp}(\text{while } E \text{ do } C, Q) = \]
\[ I \land \]
\[ \forall y.((E \land I) \to \text{wp}(C, I \land (x < y))[y/x]) \land \]
\[ \forall y.((\neg E \land I) \to Q)[y/x] \]

where \(<\) is well-founded, \(y\) represents the state before loop execution

Alternatively,

\[ \text{wp}(\text{while } E \text{ do } C, Q) = \exists k. (k \geq 0) \land P_k \]

where

\[ P_0 = \neg E \land Q \]
\[ P_{k+1} = E \land \text{wp}(C, P_k) \]

Example:

\[ \text{wp}(\text{while } n > 0 \text{ do } n := n - 1, (n == 0)) \]
\[ P_0 = \neg(n > 0) \land (n == 0) = (n == 0) \]
\[ P_1 = (n > 0) \land \text{wp}(n := n - 1, (n == 0)) = (n == 1) \]
\[ P_2 = (n > 0) \land \text{wp}(n := n - 1, (n == 1)) = (n == 2) \ldots \]
\[ \exists k. (k \geq 0) \land P_k = (n >= 0). \]
Example:

\[ \text{wp(while } n \neq 0 \text{ do } n := n - 1, (n == 0)) \]

\[ P_0 = \neg (n \neq 0) \land (n == 0) = (n == 0) \]

\[ P_1 = (n \neq 0) \land \text{wp}(n := n - 1, (n == 0)) = (n == 1) \]

\[ P_2 = (n \neq 0) \land \text{wp}(n := n - 1, (n == 1)) = (n == 2) \ldots \]

\[ \exists k. (k \geq 0) \land P_k = (n \geq 0). \]
Proving Programs Correct

Example:

\[ 1 + 3 + 5 \ldots + n = \left(\frac{n + 1}{2}\right)^2 \]

\[ \forall n. (n \geq 1) \rightarrow \sum_{i=1}^{n} (2 \times i - 1) = n^2 \]

A program for implementing this:

\[
\{ (n \geq 0) \} \\
\]

\[
i := 0; s := 0; \]

\[
\text{while } i \neq n \text{ do} \] \\
\[ i := i + 1; s := s + (2 \times i - 1); \]

\[
\{ (s == n \times n) \}
\]

\[ P_0 = ((i == n) \land (s == n \times n)) \]

\[ P_1 = ((i == n - 1) \land (s == i \times i)) \]

\[ P_2 = ((i == n - 2) \land (s == i \times i)) \ldots \]

\[ P_j = ((i == n - j) \land (s == i \times i)) \]

\[
wp(\text{while...}, (s == n \times n)) = ((i \leq n) \land (s == i \times i)) \]

\[
wp(i := 0; s := 0, ((i \leq n) \land (s == i \times i)) = ((0 \leq n) \land (s == 0)) \]
Proving Programs Correct

Computing $P_1$:

\[
P_1 = (i \neq n) \land wp(i := i + 1; s := s + 2 \times 1 - 1, (i == n) \land (s == n \times n))
\]

\[
= (i \neq n) \land wp(i := i + 1, (i == n) \land (s + 2 \times i - 1 == n \times n))
\]

\[
= (i \neq n) \land wp(i := i + 1, (i == n) \land (s == (i - 1) \times (i - 1))
\]

\[
= (i \neq n) \land (i + 1 == n) \land (s == (i + 1 - 1) \times (i + 1 - 1))
\]
Proving Programs Correct

Useful special cases:

\[ wp(C, Q) = T \quad C \text{ terminates with } Q \text{ from any initial state} \]
\[ wp(C, T) = T \quad C \text{ terminates from any initial state} \]
\[ wp(C, Q) = F \quad C \text{ produces a state where } Q \text{ is false from any initial state} \]
\[ \text{that results in } C \text{ terminating} \]
\[ wp(C, T) = F \quad C \text{ does not terminate from any initial state} \]

Other properties:

\[ \{ wp(C, Q) \} \quad C \quad \{ Q \} \]
\[ wp(C, F) = F \]
\[ wp(C, Q \land R) = (wp(C, Q) \land wp(C, R)) \]
\[ \text{if } Q \rightarrow R \text{ then } wp(C, Q) \rightarrow wp(C, R) \]
A multiply program:

\[
\{ \text{F1=F1SAVE \& F1}<2^{\text{bits}} \& F2<2^{\text{bits}} \& \text{LOW}<2^{\text{bits}} \}
\]

LDX  #bits  load the X register immediate with number bits
LDA  #0    load the A register immediate with the value 0

LOOP
ROR  F1    rotate F1 right circular through the carry flag
BCC  ZCOEF branch on carry flag clear to ZCOEF
CLC     clear the carry flag
ADC  F2    add with carry F2 to the contents of A

ZCOEF
ROR  A     rotate A right circular through the carry flag
ROR  LOW   rotate LOW right circular through the carry flag
DEX      decrement the X register by 1
BNE  LOOP  branch if X is non-zero to LOOP

\[
\{ \text{LOW + 2^{\text{bits}}*A = F1SAVE*F2} \}
\]

- A is an 8 bit accumulator - for the high bits of the multiply
- LOW is an 8 bit accumulator - for the low bits of the multiply
- Right rotation of A and then LOW is a 16 bit right rotation through both
- There are Z and C bits, and registers X, F1, F2
- ADC adds F2 and the C bit to A
Proving Programs Correct

Setup weakest preconditions:

\{ F1=F1SAVE & F1<256 & F2<256 & LOW<256 \}

LDX #bits \( S_1 \equiv [8/X]S_2() \)
LDA #0 \( S_2 \equiv [0/A]S_3() \)

LOOP ROR F1 \( S_3 \equiv [E_4/F1, E_8/C]S_4() \)
BCC ZCOEF \( S_4 \equiv \text{if } C==0 S_5() \text{ else } S_7() \)
CLC \( S_5 \equiv [0/C]S_6() \)
ADC F2 \( S_6 \equiv [E_3/A, E_7/C, E_9/Z]S_7() \)

ZCOEF ROR A \( S_7 \equiv [E_2/A, E_6/C]S_8() \)

ROR LOW \( S_8 \equiv [E_1/LOW, E_5/C]S_9(X, LOW, C) \)
DEX \( S_9(X, LOW, C) \equiv [X−1/X]S_{10}(X, LOW, C) \)
BNE LOOP \( S_{10}(X) \equiv \text{if } Z==0 S_3() \text{ else } \{LOW+256*A == F1SAVE*F2\} \)

\( Q = \{ \text{LOW} + 256*A = F1SAVE*F2 \} \)

\( E_1: \text{ LOW}/2 + C*128 \quad E_5: \text{ LOW mod 2} \)
\( E_2: \text{ A}/2 + C*128 \quad E_6: \text{ A mod 2} \)
\( E_3: \text{ A+F2+C mod 256} \quad E_7: (\text{A+F2+C})/256 \)
\( E_4: \text{ F1}/2 + C*128 \quad E_8: \text{ F1 mod 2} \)
\( E_9: \text{ A+F2+C mod 256} == 0 \)
Proving Programs Correct

Collapse non-branching instructions:

\[
\begin{aligned}
\{ & F1=F1\text{SAVE} \& F1<256 \& F2<256 \& LOW<256 \}
\end{aligned}
\]

<table>
<thead>
<tr>
<th>Instruction</th>
<th>S₁ ≡</th>
<th>S₂ ≡</th>
<th>S₃ ≡</th>
<th>S₄ ≡</th>
<th>S₅ ≡</th>
<th>S₆ ≡</th>
<th>S₇ ≡</th>
<th>S₈ ≡</th>
<th>S₉(X, LOW, C) ≡</th>
<th>S₁₀(X) ≡</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDX #bits</td>
<td>[8/X, 0/A]S₃()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDA #0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOOP ROR F1</td>
<td>S₃ ≡ [E₄/F₁, E₈/C]S₄()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCC ZCOEF</td>
<td>S₄ ≡ if C == 0 S₅() else S₇()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLC</td>
<td>S₅ ≡ [0/C]S₆()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADC F2</td>
<td>S₆ ≡ [E₃/A, E₇/C, E₉/Z]S₇()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZCOEF ROR A</td>
<td>S₇ ≡ [E₂/A, E₆/C]S₈()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROR LOW</td>
<td>S₈ ≡ [E₁/LOW, E₅/C]S₉(X, LOW, C)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEX</td>
<td>S₉(X, LOW, C) ≡ [X - 1/X]S₁₀(X, LOW, C)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BNE LOOP</td>
<td>S₁₀(X) ≡ if Z==0 S₃()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>else {LOW+256*A == F1\text{SAVE}*F2}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[Q = \{ \text{LOW} + 256*A = F1\text{SAVE}*F2 \}\]

\[\begin{aligned}
E₁: & \text{LOW}/2 + C*128 & E₅: & \text{LOW mod 2} \\
E₂: & A/2 + C*128 & E₆: & A \text{ mod 2} \\
E₃: & A+F2+C \text{ mod 256} & E₇: & (A+F2+C)/256 \\
E₄: & F1/2 + C*128 & E₈: & F1 \text{ mod 2} \\
E₉: & A+F2+C \text{ mod 256} == 0
\end{aligned}\]
Proving Programs Correct

Collapse non-branching instructions:

\{ F1=F1SAVE & F1<256 & F2<256 & LOW<256 \}

LDX  #bits \quad S_1 \equiv [8/X, 0/A]S_3()
LDA  #0

\textbf{LOOP}
ROR  F1 \quad S_3 \equiv [E_4/F1, E_8/C]S_4()
BCC  ZCOEF \quad S_4 \equiv \text{if } C == 0 \text{ } S_5() \text{ else } S_7()
CLC
ADC  F2 \quad S_6 \equiv [E_3/A, E_7/C, E_9/Z]S_7()

\textbf{ZCOEF}
ROR  A \quad S_7 \equiv [E_2/A, E_6/C]S_8()
ROR  LOW \quad S_8 \equiv [E_1/LOW, E_5/C, X - 1/X]S_{10}(X, LOW, C)
DEX

\textbf{BNE LOOP} \quad S_{10}(X) \equiv \text{if } Z==0 \text{ } S_3()
\quad \text{ else } \{LOW+256*A == F1SAVE*F2\}

Q = \{ LOW + 256*A = F1SAVE*F2 \}

E_1: \quad LOW/2 + C*128 \quad E_5: \quad \text{LOW mod 2}
E_2: \quad A/2 + C*128 \quad E_6: \quad \text{A mod 2}
E_3: \quad A+F2+C \mod 256 \quad E_7: \quad (A+F2+C)/256
E_4: \quad F1/2 + C*128 \quad E_8: \quad F1 \mod 2
E_9: \quad A+F2+C \mod 256 == 0
Proving Programs Correct

Collapse non-branching instructions:

\[
\{ \text{F1=F1SAVE & F1<256 & F2<256 & LOW<256} \}
\]

LDX #bits \( S_1 \equiv [8/X, 0/A]S_3() \)
LDA #0
LOOP ROR F1 \( S_3 \equiv [E_4/F1, E_8/C]S_4() \)
BCC ZCOEF \( S_4 \equiv \text{if } C == 0 \ S_5() \text{ else } S_7() \)
CLC \( S_5 \equiv [0/C]S_6() \)
ADC F2 \( S_6 \equiv [E_3/A, E_7/C, E_9/Z]S_7() \)
ZCOEF ROR A \( S_7 \equiv [E_2/A, E_6/C, E_1/LOW, X - 1/X]S_{10}() \)
ROR LOW
DEX
BNE LOOP \( S_{10}(X) \equiv \text{if } Z==0 \ S_3() \)
\text{else } \{ \text{LOW+256*A == F1SAVE*F2} \}

\( Q = \{ \text{LOW + 256*A = F1SAVE*F2} \} \)

\( E_1: \text{ LOW/2 + C*128} \quad E_5: \text{ LOW mod 2} \)
\( E_2: \text{ A/2 + C*128} \quad E_6: \text{ A mod 2} \)
\( E_3: \text{ A+F2+C mod 256} \quad E_7: \text{ (A+F2+C)/256} \)
\( E_4: \text{ F1/2 + C*128} \quad E_8: \text{ F1 mod 2} \)
\( E_9: \text{ A+F2+C mod 256 == 0} \)
Proving Programs Correct

Collapse non-branching instructions:

\{ F1=\text{F1SAVE} & F1<256 & F2<256 & LOW<256 \}

LDX \#\text{bits} \quad S_1 \equiv [8/X, 0/A]S_3()
LDA \#0

\begin{align*}
\text{LOOP} & & \text{ROR} & & \text{F1} & & S_3 \equiv [E_4/F1, E_8/C]S_4() \\
\text{BCC} & & \text{ZCOEF} & & & & S_4 \equiv \text{if } C == 0 \text{ } S_5() \text{ else } S_7() \\
\text{CLC} & & & & & & S_5 \equiv [0/C, E_3/A, E_7/C, E_9/Z]S_7() \\
\text{ADC} & & \text{F2} & & & & \\
\text{ZCOEF} & & \text{ROR} & & \text{A} & & S_7 \equiv [E_2/A, E_6/C, E_1/LOW, X - 1/X]S_10() \\
\text{ROR} & & \text{LOW} & & & & \\
\text{DEX} & & & & & & \\
\text{BNE} & & \text{LOOP} & & & & S_{10}(X) \equiv \text{if } Z==0 \text{ } S_3() \\
& & & & & & \text{ else } \{\text{LOW+256*A == F1SAVE*F2}\} \\
\end{align*}

Q = \{ \text{LOW + 256*A == F1SAVE*F2} \}

\begin{align*}
E_1: & & \text{LOW}/2 + C*128 & & \quad E_5: & & \text{LOW mod 2} \\
E_2: & & A/2 + C*128 & & \quad E_6: & & A \text{ mod 2} \\
E_3: & & A+F2+C \text{ mod 256} & & \quad E_7: & & (A+F2+C)/256 \\
E_4: & & F1/2 + C*128 & & \quad E_8: & & F1 \text{ mod 2} \\
E_9: & & A+F2+C \text{ mod 256} == 0 & & & & \\
\end{align*}
Proving Programs Correct

Put conditionals in proper form:

\{ F1=F1SAVE \land F1<256 \land F2<256 \land LOW<256 \}

LDX #bits \quad S_1 \equiv [8/X, 0/A]S_3()
LDA #0

LOOP ROR F1 \quad S_3 \equiv [E_4/F1, E_8/C]S_4()
BCC ZCOEF \quad S_4 \equiv \text{if } C == 0 \ S_5() \text{ else } S_7()
CLC \quad S_5 \equiv [0/C, E_3/A, E_7/C, E_9/Z]S_7()
ADC F2

ZCOEF ROR A \quad S_7 \equiv [E_2/A, E_6/C, E_1/LOW, X - 1/X]S_{10}()
ROR LOW
DEX
BNE LOOP \quad S_{10}(X) \equiv (\neg Z \land S_3()) \lor 
\quad (Z \land \{LOW + 256 \ast A == F1SAVE \ast F2\})

Q = \{ \text{LOW + 256} \ast A = F1SAVE \ast F2 \}

\begin{align*}
E_1: & \quad \text{LOW/2 + C} \ast 128 \\
E_2: & \quad A/2 + C \ast 128 \\
E_3: & \quad A + F2 + C \mod 256 \\
E_4: & \quad F1/2 + C \ast 128 \\
E_5: & \quad \text{LOW mod 2} \\
E_6: & \quad A \mod 2 \\
E_7: & \quad (A + F2 + C)/256 \\
E_8: & \quad F1 \mod 2 \\
E_9: & \quad A + F2 + C \mod 256 == 0
\end{align*}
Proving Programs Correct

Put conditionals in proper form:

\[
\{ \ F1=F1\text{SAVE} \& \ F1<256 \& \ F2<256 \& \ \text{LOW}<256 \ \}
\]

LDX #bits \quad S_1 \equiv [8/X, 0/A]S_3()
LDA #0

LOOP

ROR F1 \quad S_3 \equiv [E_4/F1, E_8/C]S_4()
BCC ZCOEF \quad S_4 \equiv (\neg C \land S_5()) \lor (C \land S_7())
CLC \quad S_5 \equiv [0/C, E_3/A, E_7/C, E_9/Z]S_7()
ADC F2

ZCOEF

ROR A \quad S_7 \equiv [E_2/A, E_6/C, E_1/\text{LOW}, X - 1/X]S_{10}()
ROR LOW
DEX

BNE LOOP \quad S_{10}(X) \equiv (\neg Z \land S_3()) \lor \\
(\neg Z \land \{\text{LOW} + 256 * A \equiv F1\text{SAVE} * F2\})

Q = \{ \ \text{LOW} + 256*A = F1\text{SAVE}*F2 \ \}

\begin{align*}
E_1: & \quad \text{LOW}/2 + C*128 & E_5: & \quad \text{LOW} \mod 2 \\
E_2: & \quad A/2 + C*128 & E_6: & \quad A \mod 2 \\
E_3: & \quad A+F2+C \mod 256 & E_7: & \quad (A+F2+C)/256 \\
E_4: & \quad F1/2 + C*128 & E_8: & \quad F1 \mod 2 \\
E_9: & \quad A+F2+C \mod 256 == 0
\end{align*}
Create recursive functions representing weakest preconditions:

\[ S_7 \equiv [E_2/A, E_6/C, E_1/LOW, x - 1/x] S_{10} \]
Create recursive functions representing weakest preconditions:

\[ S_7 \equiv [E_2/A, E_6/C, E_1/LOW, x - 1/x] (\neg Z \land S_3) \lor (Z \land \{LOW + 256 \times A = F1SAVE \times F2\}) \]
Create recursive functions representing weakest preconditions:

\[ S_7 \equiv [E_2/A, E_6/C, E_1/LOW, X - 1/X](\neg Z \land [E_4/F1, E_8/C](\neg C \land S_5) \lor (C \land S_7)) \lor (Z \land \{LOW + 256 \ast A = F1SAVE \ast F2\})) \]
Create recursive functions representing weakest preconditions:

\[ S_7 \equiv [E_2/A, E_6/C, E_1/LOW, X - 1/X](\neg Z \land [E_4/F1, E_8/C](\neg C \land [0/C, E_3/A, E_7/C, E_9/Z]S_7) \lor (C \land S_7)) \lor (Z \land \{LOW + 256 \ast A = F1SAVE \ast F2\})) \]
Proving Programs Correct

Create recursive functions representing weakest preconditions:

\[ S_7 \equiv [E_2/A, E_6/C, E_1/LOW, X - 1/X] (\neg Z \land [E_4/F_1, E_8/C] (\neg C \land [0/C, E_3/A, E_7/C, E_9/Z] S_7) \lor (C \land S_7)) \lor (Z \land \{LOW + 256 \ast A = F_1SAVE \ast F_2\}) \]

(DEFUN WP-ZCOEF (F1 C LOW A F1SAVE F2 X)
  (IF (EQUAL (DEC X) 0)
      (EQUAL (+ (* (+ (* C 128) (FLOOR A 2)) 256)
               (+ (* (MOD A 2) 128) (FLOOR LOW 2)))
               (* F1SAVE F2))
      (WP-ZCOEF
       (+ (* (MOD LOW 2) 128) (FLOOR F1 2))
       (* (MOD F1 2) (FLOOR (+ (+ (* C 128) (FLOOR A 2)) F2) 256))
       (+ (* (MOD A 2) 128) (FLOOR LOW 2))
       (+ (* (- 1 (MOD F1 2)) (+ (* C 128) (FLOOR A 2)))
        (* (MOD F1 2) (MOD (+ (+ (* C 128) (FLOOR A 2)) F2) 256)))
       F1SAVE
       F2
       (DEC X))))