Recall the Mostek multiply program discussed in class:

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

- **LDX 8** 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

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

where
- 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 shift of the 16 bit accumulator
- There are Z and C bits, and 8-bit register X
- F1 and F2 contain the input numbers to multiply
- F1SAVE is a ghost variable that retains the value of F1 throughout
- ADC adds F2 and the C bit to A
- The semantics below represent substitutions that are executed in parallel
- The semantics of ROR ANY is given by
  \[
  \text{ANY} := \text{ANY}/2 + \text{C*128}; \\
  \text{C} := \text{ANY} \% 2;
  \]
  (remember: both lines exec in parallel)
- The semantics of ADC F2 is given by
  \[
  \text{Z} := (A+F2+C)\%256 == 0; \quad (\text{integer division - the remainder}) \\
  \text{A} := (A+F2+C)/256; \\
  \text{C} := (A+F2+C)/256; \quad (\text{integer division - the quotient})
  \]
- The semantics of DEX is given by
  \[
  \text{Z} := (X-1)==0; \quad (\text{old X value}) \\
  \text{X} := (X-1)\%256; \quad (\text{range: 0-255})
  \]
- The postcondition for the multiply program is:

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

The Problem:

1. Find the weakest precondition procedure at the line labeled ZCOEF as a recursive call to the weakest precondition procedure. The following pages show how to do this for the line labeled LOOP in terms of the ZCOEF WP procedure - similar considerations apply in getting the result for the line labeled ZCOEF.

2. Acquire some common lisp for your personal computer or, on helios, use lx86cl. Download mult, 6502, and to-fcn1-ac12.lisp from the course webpage showing “lecture notes” (11 May) in the same directory. Run lx86cl. Enter (load "to-fcn1-ac12.lisp"). Enter events. Compare your result with what was obtained mechanically. Do you observe any differences?
Example calculation: finding the weakest precondition **defun** at the line labeled **LOOP** in terms of a call to the weakest precondition **defun** at the line labeled **ZCOEF**.

**Note:** the following does not do what a mechanical weakest precondition algorithm in **to-fcn1-ac12.lisp** does. That algorithm collapses contiguous instructions not involving branches into a single predicate transformer. The choice on whether to collapse these instructions depends on whether the program contains instructions (e.g. computed goto’s) that may branch into the middle of such a contiguous block. Thus, the algorithm would first collapse the straight-line segments into single “nodes” via backward substitutions (in this example that results in “nodes” at **LDX 8**, **LOOP ROR F1**, **CLC**, and **ZCOEF ROR A**.) I describe the collapsed points as “nodes” because one normally thinks of the control structure of a program as a graph (I have avoided talking about this in class because, for this simple example, doing so would cause unnecessary complications in the discussion - in my opinion). A procedure that follows what I do below could be used to derive a recursive **defun** at any desired line of the program. Hence, the assignment forces you specifically to derive the recursion at **ZCOEF ROR A**. Doing so will match the results obtained in **events** (see part 2. of the assignment above).

Label each line **W1**, **W2**, ..., **W10** from **LDX 8** through **BNE LOOP**. These labels will correspond to weakest preconditions at their corresponding lines. Thus, **W3** corresponds to the line with label **LOOP** and **W7** corresponds to the line with label **ZCOEF**. In this example we will write **W3** in terms of **W7**.

The semantics for the line **ROR F1** are:

\[ F1 := \frac{F1}{2} + C \times 128 \]
\[ C := F1 \mod 2 \]

so we can write **W3** in terms of **W4** via the two substitutions (all other parameters do not change so they are not shown):

\[ W3 := \left[ \frac{(F1 \mod 2)}{C}, \frac{(F1/2+C \times 128)}{F1} \right] W4 \]

The semantics for the line **BCC ZCOEF** are:

\[ ((1-C)==1 \land W7) \lor (C==1 \land W5) \]

(Since \( C \) takes value 0 or 1, \((1-C)\) negates the value of \( C \). We write the semantics of **BCC ZCOEF** in this way to agree with the output of the WP generator **to-fcn1-ac12.lisp**). After substituting from above:

\[ W3 := \left[ \frac{(F1 \mod 2)}{C} \right] ((1-C)==1 \land \left[ \frac{(F1/2+C \times 128)}{F1} \right] W7 \lor \]
\[ C==1 \land \left[ \frac{(F1/2+C \times 128)}{F1} \right] (0/C) W6) \]

The semantics for the line **CLC** are:

\[ C := 0; \]

so the above becomes:

\[ W3 := \left[ \frac{(F1 \mod 2)}{C} \right] ((1-C)==1 \land \left[ \frac{(F1/2+C \times 128)}{F1} \right] W7 \lor \]
\[ C==1 \land \left[ \frac{(F1/2+C \times 128)}{F1} \right] (0/C) W6) \]

The semantics for **ADC F2** after simplification due to application of pred. trans. \([0/C]\) are:

\[ C := (A+F2)/256; \quad \text{(the carry bit is cleared by \([0/C]\) so it is omitted from the sum)} \]
\[ Z := (A+F2) \mod 256 = 0; \]
\[ A := (A+F2) \mod 256; \]

so the above becomes:

\[ W3 := \left[ \frac{(F1 \mod 2)}{C} \right] ((1-C)==1 \land \left[ \frac{(F1/2+C \times 128)}{F1} \right] W7 \lor \]
\[ C==1 \land \left[ \frac{(F1/2+C \times 128)}{F1} \right] (0/C) ((A+F2)/256) W7) \]
In the above, observe only C, A, and F1 change values. But the value change for F1 is the same in both calls to W7, namely:

\[(F1/2+C*128)\rightarrow F1.\]

Observe also that the value of A changes only if \((F1\%2)\) is 1 and in that case the change is \((A+F2)\%256\rightarrow A\). This may be expressed as:

\[((F1\%2)*(A+F2)\%256 + (1-(F1\%2))*A)\rightarrow A.\]

Finally, if \((F1\%2)\) is 0 at W3, then the carry is 0 at W7 (via the first of the two ored expressions on W7 where no change is made to C) but if \((F1\%2)\) is 1 at W3 then the carry is \((A+F2)/256\) at W7 (via the second of the two ored expressions on W). This is expressed as:

\[((F1\%2)*(A+F2)/256)\rightarrow C.\]

Now the defun for the weakest precondition at the line labeled LOOP can be expressed, after rewriting the above arithmetic expressions as lisp expressions:

\[
\begin{align*}
\text{(defun W3 (F1 C LOW A F1SAVE F2 X))} \quad & \quad \text{(W7)} \\
& \quad (+ (FLOOR F1 2) (* C 128)) \\
& \quad (* (MOD F1 2) (FLOOR (+ A F2) 256)) \\
& \quad LOW \\
& \quad (+ (* (MOD F1 2) (MOD (+ A F2) 256)) (* (- 1 (MOD F1 2)) A)) \\
& \quad F1SAVE \\
& \quad F2 \\
& \quad X))
\end{align*}
\]

where the argument list for W7 is the same as the argument list for W3 - that is, F1, C, LOW, A, F1SAVE, F2, and X.