Suppose you stored an array of five values beginning at location A:

\[
\begin{align*}
A, & \quad 10; \quad 20; \quad 30; \quad 40; \quad 50 \\
C5, & \quad 5 \\
Count, & \quad 0 \\
Sum, & \quad 0
\end{align*}
\]

How could you use a loop to add them up? Assuming that C5 contains the constant 5, the size of A, will the following code work?

```
*0200
Main, cla cll  / clear AC and Link
tad C5  / load 5
    cia  / negate it
dca Count  / store -5 at Count and clear AC
Loop, tad A  / add in a
    isz Count  / loop control: 5 times
    jmp Loop
    dca Sum  / store final sum
```

Trace this code by hand or assemble and run it yourself - what does it do?

One way to fix the problem is to use “self-modifying” code. For example suppose we added the following line of code to the body of our loop. What would happen? Does this solve the problem?

```
Loop, tad A  / add in a
    isz .-1  / modify address of previous instruction
    isz Count  / loop control: 5 times
    jmp Loop
```

If you think you understand go on and write, assemble and execute a program using a loop to sum an array of 5 values using self-modifying code.

A problem with self-modifying code is that you can’t re-run it since it changes itself! Therefore we need to save the instruction at some location (allocate another variable called Temp) and restore it afterwards.

```
*0200
Main, cla cll  / clear AC and Link
tad C5  / load 5
cia  / negate it
dca Count  / store -5 at Count and clear AC
tad Loop  / get instruction
dca Temp  / & save it
Loop, tad A  / add in a
    isz .-1  / modify address of previous instruction
    isz Count  / loop control: 5 times
    jmp Loop
dca Sum  / store final sum
tad Temp  / restore instruction
dca Loop
```
A Better Way – Indirection

Begin by storing the address of A+1, the first component in the array, at location A. Load the variable \texttt{Ptr} with this address and use indirect addressing to access A (\texttt{add i Ptr}). To access the next position in array A, instead of using modifying the code, we \textit{increment} the contents of \texttt{Ptr}, which yields the address of the next component.

*0200
Main, cla cll / clear AC and Link
tad C5 / load 5
cia / negate it
dca Count / store -5 at Count and clear AC
tad A / get address of A
dca Ptr / and store it in \texttt{Ptr}
Loop, tad i Ptr / add in a – note indirection!
isz Ptr / increment address
isz Count / loop control: 5 times
jmp Loop
dca Sum / store final sum
htl
jmp Main

*0300
A, .+1; 10; 20; 30; 40; 50 / A contains address of 1st item
Ptr, 0
C5, 5
Count,0
Sum, 0

A Variant to the Above: We can also reverse the order from \textit{access and increment} to \textit{increment then access}. If done this way we store the address of A (not A+1) at location A.

*0200
Main, cla cll / clear AC and Link
tad C5 / load 5
cia / negate it
dca Count / store -5 at Count and clear AC
tad A / get address of A
dca Ptr / and store it in \texttt{Ptr}
Loop, isz Ptr / increment address
isz i Ptr / add in a – note indirection!
isz Count / loop control: 5 times
jmp Loop
dca Sum / store final sum
htl
jmp Main

*0300
A, .; 10; 20; 30; 40; 50 / A contains its own address
Ptr, 0
C5, 5
Count,0
Sum, 0

The \textit{increment then access} approach is used by the PDP-8’s auto-index indirect addressing mode.