

## Notes on “Threaded” Location Counter

(From System/370 and Assembler Language Programming, by John Ehrman)

There is an interesting aspect of multi-csect assemblies that is occasionally puzzling to the programmer. The leftmost column of the Assembler’s printed listing shows the value of the LC assigned to the code generated for each statement. (Having a multi-csect assembly listing to examine during this discussion will be helpful.) When a single control section is being assembled, these LC values increase predictably as succeeding bytes are assembled. However, in a multi-csect assembly, the values of the LC jump up and down as the control section changes. Furthermore, even though the statements belonging to different csects are interleaved, the LC values look as though the statements had been rearranged with each control section completely to itself.

The Assembler actually maintains a separate Location Counter for each distinct control section in an assembly. Except for the first csect (whose LC may be initialized to a non-zero value with a START statement), these LC values are set to zero whenever a new control section is encountered. Then, whenever a CSECT statement indicates that a different control section is to be begun or resumed, the Assembler uses the appropriate new LC to continue the assembly.

After the first pass of the assembly is complete, the Assembler processes its collection of Location Counters. First, the length of each csect is determined; this is possible because the Assembler has kept both the current and the maximum value of the LC for each control section. Then, beginning with the first csect (whose initial LC value is known), the Assembler adds the initial LC value for each control section to its length, rounds the sum up to the next doubleword boundary, and assigns the resulting value as the initial LC value of the next csect. In this way, the control sections appear to have been “unscrambled” and assembled end-to-end.

During the second assembly pass, the Assembler uses these adjusted LC values in computing the values of expressions involving symbols and Location Counter References. The relocatability attribute of the symbol is used to tell which csect the symbol belongs in, and therefore what LC value should be added to the symbol’s value found in the symbol table. (Remember that the LCs for all but the first csect are initialized to zero. In fact, the LC value given on the START statement is simply saved for use during the LC adjustment phase between passes, and the LC for that csect is also set to zero.)

This process of adjusting the LC values so that each control section starts at the end of the previous one is called threading the location counters. If it were not done, each control section would appear to start at location zero.

Why perform threading? The answer lies in the characteristics of the simple operating systems used in the early days of System/360, on which programs were not organized into complex structures by the Linkage Editor, but were loaded from object modules directly into memory. Because the address where loading began was easily determined, that address could be assigned

## Notes on “Threaded” Location Counter

(From System/370 and Assembler Language Programming, by John Ehrman)

as the initial LC value in a START statement. Thus, the LC values printed on the Assembler’s listing corresponded exactly to the addresses in memory occupied by the assembled and loaded code. This made debugging simpler, since an interruption at some address could be immediately identified with the offending instruction.

In effect, threading is intended simply as a convenience to the programmer. We will see in discussing linkage editing that the LCs must be “un-threaded” by the Linkage Editor so that it can correctly relocate the program. Whether threading is a help or hindrance is a matter of personal preference.