This processor has no stack (insert spooky laughter)

devblogs.microsoft.com/oldnewthing/20161031-00

October 31, 2016

Raymond Chen

In the early days of computing, most processors did not have what we today would consider a “stack”. Everything was done with static data.

Subroutine linkage on these systems was typically done by explicitly passing the return address to the called subroutine. The first thing the subroutine did was save this return address in a global variable somewhere, so it knew where to go when the subroutine finished.

One convention was to use self-modifying code and store the return address directly in the branch target of a jump instruction at the end of the function.

Another convention was to store the return address in a global variable somewhere, and then perform an indirect jump at the end of the function. A common place for this global variable was immediately before the first instruction of the subroutine. Indeed, some processors elevated this convention into hardware: The “Jump to subroutine” instruction automatically stored the return address at the specified target subroutine address, and then resumed execution at the word immediately following the return address. In other words every subroutine looked like this:

```
MYSUBROUTINE:
    .WORD 0             ; return address goes here
    blah blah           ; first actual instruction
    blah blah           ; do stuff
    JMP @MYSUBROUTINE   ; indirect jump back to caller
```

This subroutine linkage convention precluded recursion, because a recursive call would destroy the previous activation frame’s return address. (It also precluded multithreading, but that’s way beyond where we are at this point in history.)

The lack of a stack also means that nothing has automatic storage duration. Function local variables actually have static storage duration.

This is the “fast one” that I pulled last week when I discussed the COMMON statement in FORTRAN. I put local variables into a COMMON block and noted that they were being shared across functions. This implied that the local variables were static, because if they were automatic, then there would be nothing left to share once the function returned.

Raymond Chen

Follow