Determining the Length of Literal Strings for C Functions

C strings have no soul, is something I like to tell my students. For example, Unlike C++ strings, you can't ask them to tell you about themselves. Instead you must use other functions such as `strlen()`.

When working with APIs that use C strings, you must often tell the API the length of those strings (because, well, C string have no soul). Here is an example:

```
write(1, "Hello, World!\n", 14);
```

This sends the familiar string to `stdout`.

In Assembly Language

When a string is fixed within your assembly code, you can let the assembler itself calculate the length for you.

In assembly language, the string would likely have been placed in a `.data` section using the `.asciz` directive. But! How to get the length of the string? You could:

- hard code the length as I did above, or
- go through the effort of calling `strlen()`

There is a third option as demonstrated by the following program:

```
        .global         main                                            // 1
        .align          2                                               // 2
        .text                                                           // 3
                                                                        // 4
main:   str             x30, [sp, -16]!                                 // 5
        mov             w0, 1               // stdout                   // 6
        ldr             x1, =s              // pointer to string        // 7
        ldr             x2, =ssize          // pointer to computed length // 8
        ldr             w2, [x2]            // actual length of string  // 9
        bl              write                                           // 10
                                                                        // 11
        ldr             x0, =fmt                                        // 12
        ldr             x1, =s                                          // 13
        ldr             x2, =ssize                                      // 14
        ldr             w2, [x2]                                        // 15
        bl              printf                                          // 16
                                                                        // 17
        ldr             x30, [sp], 16                                   // 18
        mov             w0, wzr                                         // 19
        ret                                                             // 20
                                                                        // 21
```

```
        .data                                                           // 22
                                                                        // 23
s:      .asciz          "Hello, World!\n"                               // 24
ssize:  .word           ssize - s - 1       // accounts for null at end // 25
fmt:    .asciz          "str: %slen: %d\n"  // accounts for newline     // 26
                                                                        // 27
        .end                                                            // 28
```

`Line 24` contains the string. It is null terminated.

`Line 25` is the new learning. The assembler calculates the difference between the address of `s` and the address of `ssize` and puts it at the location of `ssize`. One is subtracted from the length because of the null termination of the string. You might not see the null terminator but it takes up space.

An example of using the stored length is on `Lines 8` and `9` like any other statically stored data.

Here is the output of the program:

```
Hello, World!
str: Hello, World!
len: 14
```