Section 1 / Defining `structs`

Given:

```
struct Foo {
    short a;
    char b;
    int c;
};

struct Foo Bar = { 0xaaaa, 0xbb, 0xcccccccc };
```

Here is one way of defining and accessing the struct:

```
#include "apple-linux-convergence.S"                                    // 1
                                                                        // 2
        GLABEL        main                                              // 3
        .text                                                           // 4
        .p2align    2                                                   // 5
                                                                        // 6
MAIN                                                                    // 7
        PUSH_P      x29, x30                                            // 8
        mov         x29, sp                                             // 9
                                                                        // 10
        LLD_ADDR    x0, fmt                                             // 11
        LLD_ADDR    x1, bar                                             // 12
        ldrh        w2, [x1, 0]                                         // 13
        ldrb        w3, [x1, 2]                                         // 14
        ldr         w4, [x1, 4]                                         // 15
#if defined(__APPLE__)                                                  // 16
        PUSH_P      x3, x4                                              // 17
        PUSH_P      x1, x2                                              // 18
        CRT         printf                                              // 19
        add         sp, sp, 32                                          // 20
#else                                                                   // 21
        CRT         printf                                              // 22
#endif                                                                  // 23
        POP_P       x29, x30                                            // 24
        mov          w0, wzr                                            // 25
        ret                                                             // 26
                                                                        // 27
        .data                                                           // 28
                                                                        // 29
fmt:    .asciz      "%p a: 0x%lx b: %x c: %x\n"                         // 30
bar:    .short      0xaaaa                                              // 31
        .byte       0xbb                                                // 32
        .byte       0               // padding                          // 33
```

```
        .word       0xcccccccc                              // 34
                                                            // 35
        .end                                                // 36
```

It would be understandable if you don't see where the structure of the `struct` is being specified in the code. That's because it isn't. Rather, focus on the 0, 2 and 4 on lines 30 through 32. These are the hard coded offsets of the struct's fields `a`, `b` and `c`.

A second way to define the offsets of the fields within a struct which is preferable to the one above is excerpted here.

The full text of the file is located here.

```
        .equ        foo_a, 0  // like #define
        .equ        foo_b, 2  // like #define
        .equ        foo_c, 4  // like #define
```

and here:

```
        ldrh        w2, [x1, foo_a]
        ldrb        w3, [x1, foo_b]
        ldr         w4, [x1, foo_c]
```

This method uses `.equ` to make the offsets into symbolic constants. This is just like using `#define` in C and C++. That is, the above is equivalent to the following in C or C++:

```
#define foo_a 0
#define foo_b 2
#define foo_c 4
```

Finally, here is a third way of defining `structs`. However, this method works on Linux but not on Apple. We have not yet discovered the incantation that allows something like this on Apple.

The salient Linux-only code is excerpted below:

```
        .section    Foo
        .struct     0           // a starts at 0 and goes for 2
Foo.a:  .struct     Foo.a + 2   // b starts at 2 and goes for 2
Foo.b:  .struct     Foo.b + 2   // c starts at 4
Foo.c:
```

This method has a *substantial* benefit over the previous methods. Imagine you need to insert a new field between `Foo.a` and `Foo.b`. Simply do so. If you're using this third method, which is based on relative offsets, the assembler will do the work of adjusting the following offsets for you.