Section 1 / Defining `structs`

Given:

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

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

Here is one way of defining and accessing the struct:

```
        .global         main                                    // 1
        .text                                                   // 2
        .align          2                                       // 3
                                                                // 4
main:                                                           // 5
        str             x30, [sp, 16]!                          // 6
                                                                // 7
        ldr             x0, =fmt                                // 8
        ldr             x1, =Bar                                // 9
        ldrh            w2, [x1]                                // 10
        ldrb            w3, [x1, 2]                             // 11
        ldr             w4, [x1, 4]                             // 12
        bl              printf                                  // 13
                                                                // 14
        ldr             x30, [sp], 16                           // 15
        mov             w0, wzr                                 // 16
        ret                                                     // 17
                                                                // 18
        .data                                                   // 19
                                                                // 20
fmt:    .asciz          "%p a: 0x%x b: 0x%x c: 0x%x\n"          // 21
```

It would be understandable if you don't see where the `struct` is being defined. That's because it isn't. Rather, the implied +0 on `line 10` and the 2 and 4 on `lines 11` and `12` are the hard coded offsets into the `struct`.

Here is a second way to define a `struct`.

```
        .global         main                                    // 1
        .text                                                   // 2
        .align          2                                       // 3
                                                                // 4
        .equ            foo_a, 0        # like #define          // 5
        .equ            foo_b, 2        # like #define          // 6
        .equ            foo_c, 4        # like #define          // 7
```

```
                                                                    // 8
main:                                                               // 9
        str             x30, [sp, 16]!                              // 10
                                                                    // 11
        ldr             x0, =fmt                                    // 12
        ldr             x1, =Bar                                    // 13
        ldrh            w2, [x1, foo_a]                             // 14
        ldrb            w3, [x1, foo_b]                             // 15
        ldr             w4, [x1, foo_c]                             // 16
        bl              printf                                      // 17
                                                                    // 18
        ldr             x30, [sp], 16                               // 19
        mov             w0, wzr                                     // 20
        ret                                                         // 21
                                                                    // 22
        .data                                                       // 23
                                                                    // 24
fmt:    .asciz          "%p a: 0x%x b: 0x%x c: 0x%x\n"              // 25
```

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`.

```
        .global         main                                        // 1
        .text                                                       // 2
        .align          2                                           // 3
                                                                    // 4
main:                                                               // 5
        str             x30, [sp, 16]!                              // 6
                                                                    // 7
        ldr             x0, =fmt                                    // 8
        ldr             x1, =Bar                                    // 9
        ldrh            w2, [x1, Foo.a]                             // 10
        ldrb            w3, [x1, Foo.b]                             // 11
        ldr             w4, [x1, Foo.c]                             // 12
        bl              printf                                      // 13
                                                                    // 14
        ldr             x30, [sp], 16                               // 15
        mov             w0, wzr                                     // 16
        ret                                                         // 17
                                                                    // 18
        .section        Foo                                         // 19
```

```
        .struct         0           // a starts at 0 and goes for 2     // 20
Foo.a:  .struct         Foo.a + 2   // b starts at 2 and goes for 2     // 21
Foo.b:  .struct         Foo.b + 2   // c starts at 4                    // 22
Foo.c:                                                                  // 23
                                                                        // 24
        .data                                                           // 25
                                                                        // 26
fmt:    .asciz        "%p a: 0x%x b: 0x%x c: 0x%x\n"                    // 27
```

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.