Using `ldw, ldh, ldb, stw, sth, stb, .hword, .byte`

We've already been introduced to the **executable instructions** `ldw` and `stw`, which are used to load words from memory into a register, or store words into memory form a register.

But what if you want to store data in memory that isn't the size of a whole word (4 bytes)? Because the processor is byte-addressable, this is possible.

Recall that the line `ldw r9, (r10)` will load 4 bytes of data from the address given by `r10` in memory into register `r9`.

Similarly we can also load a **half-word** or a **single byte** into a register using `ldh` and `ldb` respectively instead of `ldw`.

We can also store half-words and single bytes into memory using `sth` and `stb` instead of `stw`. Using `stb` and `sth` will always use the least significant byte(s) in the register.

In addition to these executable instructions, we also previously looked at an **assembler directive** to store a word in memory: `.word`.

For example, the line

```
answer: .word 0
```

will load 0 into the memory address labeled by `answer`.

Again, what if we want to load into memory a half-word, or a single byte? There is a solution for this. We can use `.byte` to store a byte at the next address in sequence and we can use `.hword` to store 2 bytes at the next address in sequence.

However here, we might run into some issues.

Using `.align`, `.skip`

There are certain rules about storing words and half-words in memory. The most important of which: the addresses words **must** be divisible by 4, and the addresses of half-words **must** be divisible by 2.

That is, you can store a word at locations `0x0, 0x4, 0x8` etc. but you cannot store a word at `0x3` or `0x1`.

To understand this, lets take a look at the following assembly code:

```
.global _start
_start:
```

```
        movi r8, meByte
        ldb r9, (r8)
        movi r8, meWord
        ldw r9, (r8)
done: br done

meByte: .byte 55
meWord: .word 0x2143032c
```

The above code will not assemble. Why? Let's take a look at how these instructions would be stored in memory.

memory address	instruction/ directive
0x0	movi r8, meByte
0x4	ldb r9, (r8)
0x8	movi r8, meWord
0xC	ldw r9, (r8)
0x10	done: br done
0x14	meByte: .byte 55
0x15	meWord: .word 0x2143032c

The `.byte` directive only uses a single byte, and in the example above, is at memory address `0x14`. Since it only uses a single byte, the next address in sequence is `0x15`. However, since `0x15` is not divisible by 4 (or 2), we cannot store a word there (or a half-word).

There are two assembler directives to solve this: `.align` and `.skip`.

`.align` takes in one argument `n`, used like `align 2` for example and will make sure the next address in sequence is divisible by 2^n. So, for example, `align 2` would make sure the next address is divisible by 4 which would let you store a word in that address.

`.skip` also takes in one argument `n`, used like `skip 3`, and will make sure the next address in sequence is `n` addresses after the current one.

The above code can therefore be re-written like this:

```
.global _start
_start:
        movi r8, meByte
        ldb r9, (r8)
        movi r8, meWord
        ldw r9, (r8)
done: br done

meByte: .byte 55
```

```
                .align 2 /* or .skip 3 */
meWord: .word 0x2143032c
```