Translation Table

- I missed a bit here: lecture 15 and possibly others

The ASCII table allows us to translate betweens symbols and numbers.

To translate between numbers and the 7-segment display, for example, we need to create an appropriate translation table, that tells us which LEDs need to be turned on to display each value.

1 bit controls each LED, and the 8th bit controls the enable switches of the two digit displays, i.e. if the 8th bit is a zero, the left-hand digit is enabled and the right-hand one is disabled.

In Samphire, when you disable a digit, it continues displaying whatever value it was last displaying while enabled. In the real world, you swap between the two digits fast enough that it looks like they're both always on, taking advantage of a property of human vision.

Building the Table

0

We need LEDs a, b, c, d, e, g to be = 1 and LED f to be = 0. Then the value of h will determine which digit is displayed on.

So we need either `11111010` (for the left-hand digit) or `11111011` (for the right-hand digit), equivalent to FA and FB respectively.

1

We just want LEDs g and e. This gives us `00001010` and `00001011`, equivalent to `0A` and `0B`.

2

`10110110` and `10110111`, equivalent to B6 and B7.

Complete Table

Display	LHS	RHS
0	FA	FB
1	0A	0B
2	B6	B7
3	9E	9F
4	4E	4F
5	DC	DD
6	FC	FD
7	8A	8B
8	FE	FF
9	CE	CF

Note that the displays always differ by 1, since it's the least significant bit that controls which display.

Creating the Table in Samphire

In Samphire we can use the Assembler Directive[1] DB (Define Byte) along with ORG to place specific bytes into memory, starting at a specific memory location.

The following code will place the byte FA into memory at address 50:

```
org 50
db fa
```

Each subsequent DB directive will be placed into the next following memory location.

We can now create our table:

```
org 50

db fa
db 0a
db b6
db 9e
db 4e
db dc
db fc
db 8a
db fe
db ce
```

The translation table (for the LHS display) is now achieved by using the memory locations 50 to 59. We don't need to store a table for the RHS digit, as we can just add 1 to the value for the corresponding LHS digit. E.g. to get '1' on the RHS display we add 1 to the value 0A to get 0B.

Using the Table

We want to take in a single digit from the keyboard, and then display it on the 7-digit display. To do this, we subtract 30 (to convert from ASCII) and add 50 (to get to the table in our example), or add 20:

```
in 00
add al, 20
mov al, [al]
out 02
```

Once we've taken the input, we want to output the value at the memory address stored in al. We do this by moving that value (FA if we input 0) from the address it's in (50, currently in the al register) into the al register itself, using `mov al, [al]`.

If we want to output to the other digit, we simply `inc al` before outputting to port 2.

The following code will continuously count from 0 to 9 on the RHS display:

```
outer_loop:
    mov bl, 50
    inner_loop:
        mov al, [bl]
        inc al
        out 02
        inc bl
        cmp bl, 5a
        jnz inner_loop
    jmp outer_loop

org 50

db fa
db 0a
db b6
db 9e
db 4e
db dc
db fc
db 8a
db fe
db ce
```

Class Exercise

See if you can get it to count from 00 to 99 in decimal continuously.

---

1. Assembler directives are instructions for the assembler, and are executed when the code is assembled, as opposed to instructions, which are executed at program runtime. They're not placed into memory by the assembler.↩