Computer Systems Week 8 Lab

Daniel Coady (102084174)

01/10/2019

The error

I have changed the line

```
mov r2,$0F0000
```

To use the first 6 numbers of my student number (102084 or 0x18EC4)

```
flat assembler for ARM  version 1.43 (built on fasm 1.73.02)
(16384 kilobytes memory)
OK2.ASM [17]:
   mov r2,#102084
processed: mov r2,#102084
error: Immediate value cannot be encoded.
```

The mov instruction

Why does mov only work with numbers that have 24 bits set to 0?

The mov instruction uses 20 of 32 bits for the opcode and 4 bits for the ROR. This allows the 32-bit word to contain the full instruction of what needs to happen, but essentially means you are limited to 8-bits to write to while the rest have to be consecutive zeros. It is worth mentioning however that the zeroes only need to be consecutive once rotated, so values of 0xFF0000 and 0x00FF00 are valid because once rotated the zeroes will be consecutive, allowing for the storing of the opcode and ROR amount.

How can mov still be used for numbers that do not satisfy this?

As previously mentioned we can use ROR to handle rotating these bits around to make room for the opcode to fit in. This is achieved using the barrel shifter which is a hardware shift register that loops back on itself so that when bits are pushed out one end, they are then pushed onto the other end of the register.

Identify the three bytes (as hex digits) needed to construct your student number, and write the code to load the entire number into a register

The three bytes that would need to be written to construct my student number are 0x18000, 0x00EC0, and 0x00004. To put this into a register you could do the following:

```
mov r0, $18000
orr r0, $00EC0
orr r0, $00004
```

Flashing 3 times

Getting the LED to flash 3 times before a pause was a fairly trivial task. All I did to achieve this was store the amount of loops we want to make in a register:

```
mov r2, #3 ; we want to loop 3 times
```

Then we have a loop that encompasses the entire flashing part of the code which subtracts one from the register each loop round, checking if it has reached 0 each time:

```
flashingloop:
  ; do the flashing stuff
  sub r2, #1
  cmp r2, #0
bne flashingloop ; will loop back if r2 != 0
```

And then finally we will have one longer timer that will pause before we loop back on this all again.