From last time

Explain the difference between direct and indirect addressing. (2 marks)

Direct: address in instruction; Indirect: address in register

Why would it be difficult to use direct addressing in ARM Load (LDR) and Store (STR) instructions? (2 marks)

Not enough bits in instruction for all possible addresses

Initially, R1=0x11aa, R2=0x22bb, R3=0x33cc, R4=0x44dd, R5=0x1000

– what value is in R5 after each instruction?
– what values are stored where? (4 marks)

\[
\begin{align*}
\text{STR R1, } [\text{R5}] & \quad [0x1000] \leftarrow 0x11aa, \quad R5 \leftarrow 0x1000 \\
\text{STR R2, } [\text{R5, #4}] & \quad [0x1004] \leftarrow 0x22bb, \quad R5 \leftarrow 0x1000 \\
\text{STR R3, } [\text{R5, #4}] & \quad [0x1000] \leftarrow 0x33cc, \quad R5 \leftarrow 0x1004 \\
\text{STR R4, } [\text{R5, #4}] & \quad [0x1008] \leftarrow 0x44dd, \quad R5 \leftarrow 0x1008
\end{align*}
\]
Question: rewrite to use conditional instructions

(e.g. SUB → SUBGT)

so don’t increment and then decrement i (R2) at the end of the loop

```
ADRL R1, message
MOV R2, #0
again LDRB R0, [R1,R2]
CMP R0, #0
ADDNE R2, R2, #1
BNE again
STR R2, length
```
Question: use post-indexed and save a register

(e.g. STR R3, [R4], #1)

(but not using conditional instructions)

ADRL R1, message
again LDRB R0, [R1], #1
CMP R0, #0
BNE again
ADRL R0, message
SUB R0, R1, R0
SUB R0, R0, #1
STR R0, length

1 fewer instruction in loop body, several extra afterwards
Question: what if character isn’t in string?

String.indexOf() returns –1 if it gets to the end of the string without finding the character

```
ADRL R1, message
again LDRB R0, [R1],#1
CMP R0, #0
BEQ oops
CMP R0, #'e'
BNE again
ADRL R0, message
oops SUB R0, R1, R0
oops SUB R0, R0, #1
STR R0, find
```
Questions:

Why two right shifts but only one left shift?

**LSR** performs an unsigned shift right  
**ASR** performs a signed shift (two’s complement interpretation)

Assuming no overflow left shifts are the same,  
thus **ASL** $\equiv$ **LSL**

Why no *rotate* left?

ARM can shift/rotate words by any number of places; rotation is *cyclic* over 32 bits so (e.g.) **ROR** #12 $\equiv$ **ROL** #20 etc.
Question: What does this code do?

assume R0 = bottom 3 bits of an xterm colour-code

ADRL R1, table
ADD R0, R0, R0, LSL #1  \quad R0 = R0*3
MOV R0, R0, LSL #2  \quad R0 = R0*4
LDR R0, [R1, R0]!  \quad R1 = R1+R0; R0 = [R1]
SVC 4  \quad print integer in R0
ADD R0, R1, #4
SVC 3  \quad print string (address in R0)

prints the RGB colour-code (in decimal, unfortunately)
and then the colour-name