# Solutions: Tutorial 2

**Question 1:**
a)
- Not necessary to allocate an additional word in flash (saves memory)
- MOVS is one cycle, LDR is two.

b) When the data to be loaded is one byte (0 to 255)
c) Creating a breakpoint sets the contents of a CPU breakpoint register to hold the address of the instruction you want to break on. That when the PC equals the value in a breakpoint register, the CPU halts execution.
d) The external oscillator has a much tighter tolerance. Useful for timing critical applications.

**Question 2**
a) When nothing is connected to the pin (floating), the pin may pick up noise and toggle randomly between high and low. A pull resistor sets a default level when nothing else is asserting a logic level on the pin.
b) Pull up. The push buttons pull the line to ground when pressed. We want the default state (when the button is not pressed) to be the opposite of when it is pressed.
c) The SWD data and clock pins are PA13 and PA14. By default, these pins are configured as alternate mode to allow them to function as SWD pins, rather than just inputs. If 0 is written to the GPIOA_MODER, all GPIOA pins will become inputs and PA13 and PA14 will no longer function as SWD pins. Hence the debugger will not be able to communicate with the microcontroller.
d)
```
LDR R0, ADDRESS
LDR R1, [R0]
LDR R2, MASK_UPPER_OUT
ANDS R1, R1, R2
LDR R2, UPPER_AB
ORRS R1, R1, R2
STR R1, [R0]
.align
ADDRESS: .word 0x20000100
MASK_UPPER_OUT: .word 0x00FFFFFF
UPPER_AB: .word 0xAB000000
```

**Question 3**
a) LDR: 2 ; MOVS: 1,  BEQ: 3, because BEQ is taken when Z=1 and the MOVS instruction set Z=1 by moving the number 0.
Total: 6
b) As all data accesses are aligned, each instruction is on an address which is a multiple of 2 bytes. This means the lsb of the PC is always 0. We can hence assign another use to this bit. The use is to specify the instruction set of the instruction being pointed to. 1 = Thumb, 0 = ARM. Our CPU uses thumb, so this bit should be 1.

**Question 4:**
Let LDR be at address 0.
ADDS is at address 2
B is at address 4
xyz is at address 8 (align forces it to be divisible by 4; 6 would be invalid)
adc is at address 12
foo is at address 16

Hence the difference is 16. As a result of our pipeline, the PC is two halfwords ahead (at address 4) when LDR is executed, so actual offset is 16-4 = 12

b) No change. Reason: as per A6.7.27, the PC is aligned (has its lower 2 bits set to 0) when being used as the base address in LDR. Hence, although the PC will actually be at address 6 when LDR is executed, it will be aligned to 4 and the offset will be the same. The literals stay on the same addresses because they will be aligned by default. (this question was tricker than I expected…)