Solutions 5

Q1

a) The data in the HardFault vector is fetched and moved into the PC. Chances are that the data in the vector point to invalid memory which causes a second HardFault. This causes lockup.

b) The data in the HardFault vector is fetched and moved to into the PC. This causes the PC to point to a valid instruction. Execution continues from this instruction.

Note that in both of the above cases the system state is pushed to the stack prior to the vector being fetched.

c) The address of a valid instruction which we want to execute when the exception occurs.

Q2

a) It allows us an easy-to-use interface to place and retrieve data to and from RAM without worrying about what memory address the data goes to.

b) We want to give it as much space to grow so that lots of data can be placed onto the stack before collisions occur.

c) It's not a problem. Reason: the SP address is decremented BEFORE the data is written. Hence when the decrement takes place the SP will point to valid memory.
(mark for reason)

d)

```
SUB SP, SP, #4
STR R0, [SP]
SUB SP, SP, #12
STR R1, [SP, #0]
STR R2, [SP, #4]
MOV R3, LR          @ STR can only access R0-R7. Need to get LR into a high register
STR R3, [SP, #8]
LDR R0, [SP]
ADD SP, SP, #4
```

Q3

a) A block of code which can be branched to, executed, and then branches back to where it was called from

b) They maximize code reuse / minimize code duplication. This reduces code size. Also, if we

want to modify some code we only have to do it in once place. Also we don't have to maintain so many labels.

c) 0x0800 1005 This is +4 to point to next instruction and +1 to set Thumb

Q5

a) Breakpoints provide us the ability to cause the CPU to halt execution when it is about to execute the instruction at a specific address.
This allows us to cause the CPU to free-run over very long blocks of code and stop when we get to a line we're interested, instead of having to step many, many times through a large block to get to a line we're interested in.

b)
b foobar
b 94
b +7
b *0x080000F0

c) delete