```
        AREA question, CODE, READONLY
        ENTRY

        ; variables

        MOV r1, #x                      ;put param x into r1
        MOV r2, #n                      ;put param n into r2

        ; stack

        ADR r13, stack                  ;init. the stack

        STR r1,[r13,#-FOUR]!                    ;Push param x on the stack
        STR r2,[r13,#-FOUR]!                    ;Push param n on the stack

        SUB r13, #FOUR                  ; reserve a slot on stack for value we return
        BL power                        ; call subroutine 'power'

        LDR r3, =result                 ; find address of result variable
        LDR r4, [r13], #TWELVE      ; clean stack
        STR r4, [r3]                    ; store final result in result variable, r4

loop    B loop                          ; loop when done

;-------------------------------------------------------------------------------

ret             MOV r13, r11                    ; return statement – collapse stack, working spaces
                LDMFD r13!, {r1,r2,r11,r15}   ; load registers, control flow goes back to command that called the subroutine

power   STMFD r13!, {r1,r2,r11,r14}    ; push registers onto stack
                MOV r11, r13                    ; get the stack pointer pointing to top of stack
                SUB r13, #FOUR                  ; creating space for stack frame
                LDR r2, [r11, #TFOUR]           ; get parameter n from the stack
                TEQ r2, #ZERO                   ; check if x ==0
                MOVEQ r2, #ONE                  ; if it's equal to 0, then we have to return one
                                                        ; so we store one into r2
                STREQ r2, [r11, #SIXTEEN]  ; if they're equal, we move the frame pointer accordingly, getting the fp to point to the value in r2 - offset of 16 because the stack will have r0, r1, r11 (fp), r14 (lr) on the stack before it - so we skip those four slots to get to the return value register
                BEQ ret                                 ; go to return statement

                LDR r1, [r11, #TWENTY]          ; if x!=0, load
                ANDS r2, #ONE                   ; is n odd or not? the s sets a flag so we avoid an extra teq operation here
                MOVEQ r5, #ONE                  ; store 1 into r5 to keep track of if n is even or odd -- if even, we move 1 into r5
                SUBNE r2, #ONE                  ; if not equal - i.e. if n is odd - decrement by 1
                MOVEQ r2, r2, LSR #1            ; divide by two y = power(x, n>>1) => this is n>>1 => could make it asr
                STMFD r13!, {r1,r2}             ; loading new params back on the stack for next recursive call
```

```
                SUB r13, #FOUR                          ; create new stack frame
                BL power                                ; calls itself again

                TEQ r5, #ONE                            ; if r5 has 1 stored in it, n is even
                BEQ ev                                          ; so if n is even, go the 'ev' branch of code
                LDR r2,[r11,#-SIXTEEN]          ; see line 33 -- moving the frame pointer - in this case other direction
                MUL r2,r1,r2                            ; x*power(x, n-1)
                STR r2, [r11, #SIXTEEN]         ; see line 33
                B ret                                           ; now go to return statement

ev              LDR r1,[r11,#-SIXTEEN]          ; if even ... => getting value returned from prevous call x*power(x, n-1)
                STR r1,[r11,#-FOUR]                     ; y slot on stack <- returned value
                MUL r2,r1,r1                            ; y*y
                STR r2, [r11, #SIXTEEN]       ; see line 33
                B ret                                           ; go to return statement

;-------------------------------------------------------------------------------

                AREA question, DATA, READWRITE

                SPACE 0xFF   ; space for stack
stack   DCD 0           ; stack pointer
result  DCD 0                   ; result variable label

FOUR  EQU 4
ONE             EQU 1
ZERO  EQU 0
TWELVE          EQU 12
SIXTEEN         EQU    16
TFOUR           EQU 24
TWENTY          EQU 20

; parameters
x               EQU 4
n               EQU 2


                END
```

Stack frame sketch:

local variables	
r1	←r13 (stack pointer)
r2	← r11 (frame pointer)
fp	
lr	
value to return	
x	
n	

How many stack frames are needed to calculate x^n, when n = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, and 12?

n	stack frames to calculate x^n
0	1
1	2
2	3
3	4
4	4
5	5
6	5
7	6
8	5
9	6
10	6
11	7
12	6