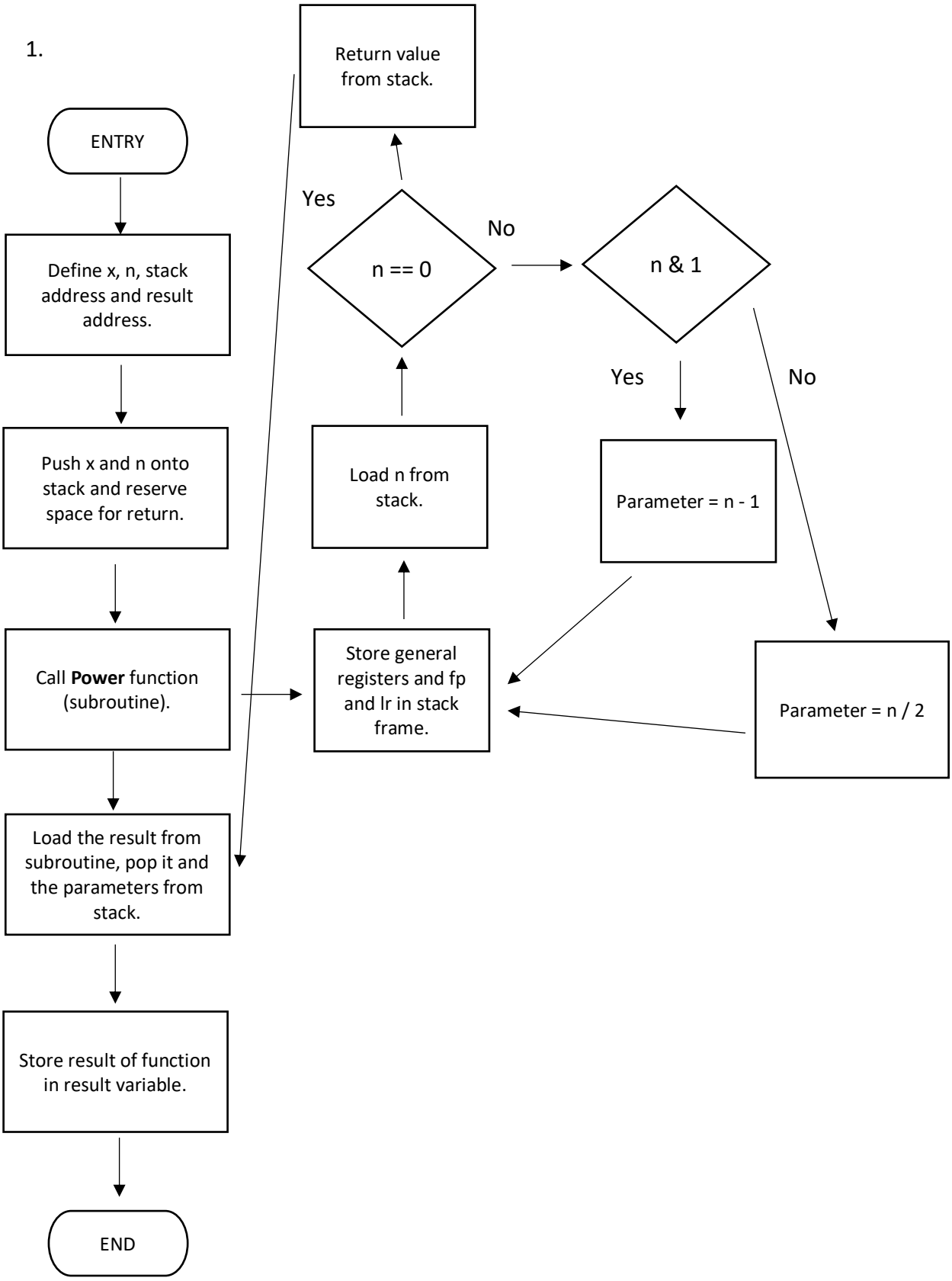
1.


ENTRY
Define x, n, stack address and result address.
Push x and n onto stack and reserve space for return.
Call **Power** function (subroutine).
Load the result from subroutine, pop it and the parameters from stack.
Store result of function in result variable.
END
Store general registers and fp and lr in stack frame.
Load n from stack.
n == 0
Yes
No
Return value from stack.
n & 1
Yes
No
Parameter = n - 1
Parameter = n / 2

Question →

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 = 0 - 1
n = 1 - 2
n = 2 - 3
n = 3 - 4
n = 4 - 4
n = 5 - 5
n = 6 - 5
n = 7- 5
n = 8 - 5
n = 9 - 6
n = 10 - 6
n = 11 - 7
n = 12 - 6

Code →

```
;-------------------------------------------------------
                AREA power, CODE, READONLY
n               EQU 2
x               EQU 3
                ENTRY
Main            ADR         sp, stack           ;define the stack
                MOV         r0, #x              ;prepare x parameter
                STR         r0, [sp, #-4]!      ;push x parameter onto stack
                MOV         r0, #n              ;prepare n parameter
                STR         r0, [sp, #-4]!      ;push n parameter onto stack
                SUB         sp, sp, #4          ;reserve a place in stack for return value
                BL          Powr                ;calls Powr subroutine (power function)
                LDR         r0, [sp], #4        ;load result in register r0 and pop from
stack
                ADD         sp, sp, #8          ;remove parameters from stack
                ADR         r1, result          ;get adress of return variable
                STR         r0, [r1]            ;store final result in result variable
Loop            B           Loop
;-------------------------------------------------------
                AREA power, CODE, READONLY
Powr            STMFD       sp!, {r0,r1,r2,fp,lr}   ;push general registers as well as fp and lr
                MOV         fp, sp              ;set fp for this call
```

```
                SUB         sp, sp, #4          ;space for local variable y
                LDR         r0, [fp, #0x1C]     ;gets parameter x from stack
                LDR         r1, [fp, #0x18]     ;gets parameter n from stack

                CMP         r1,     #0          ;if (n==0)
                BNE         Chodd               ;if not, branch to check if (n & 1)
                MOVEQ       r0, #1              ;if equal, prepare 1 to be returned
                STREQ       r0, [fp, #0x14]     ;store returned value (1) in stack
                BLE         Ret                 ;branch to return
                STR         r0, [sp, #-4]!      ;push parameter onto stack

Chodd           TST         r1, #1              ;if (n & 1) -- if odd
                BEQ         Even                ;if not, branch to even evaluation
                BNE         Odd                 ;if is, branch to odd evaluation

Odd             SUB         r1, r1, #1          ;prepare new value -- (n-1)
                STR         r1, [sp, #0]        ;push parameter onto stack
                SUB         sp, sp, #4          ;reserve a place in stack for return value
                BL          Powr                ;recursively branch on power function with
new value
                LDR         r2, [sp], #4        ;load result and pop from stack
                ADD         sp, sp, #8          ;remove parameter from stack
                MUL         r0, r2, r0          ;prepare value to be returned by mutiplying
recurssivley found value by x
                STR         r0, [fp, #0x14]     ;store returned value in stack
                B           Ret

Even            LSR         r1, r1, #1          ;evaluates (n >> 1), basically dividing by 2
                STR         r1, [sp, #0]        ;push the parameter onto the stack
                SUB         sp, sp, #4          ;reserve a place in stack for return value
                BL          Powr                ;recursively branch on power function with
new value
                LDR         r2, [sp], #4        ;load the result and pop it from the stack
                ADD         sp, sp, #8          ;remove the parameter from the stack
                STR         r2, [fp, #-4]       ;updates the value of variable y
                MUL         r0, r2, r2          ;multiplies y by y to evaluate return value
                STR         r0, [fp, #0x14]     ;stores the return value in the stack
                B           Ret

Ret             MOV         sp, fp              ;collapses all working spaces for this
function call
                LDMFD       sp!,{r0,r1,r2,fp,pc}    ;load all regiosters and return to the caller
;-----------------------------------------------------
                AREA power, DATA, READWRITE
```

```
result   DCD             0x00                    ;final result
                 SPACE   0xB4                    ;declare space for the stack
stack    DCD             0x00                    ;initial stack position
;-------------------------------------------------------
                 END
```