AREA factorial, CODE, READONLY

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

x EQU 6 ;defines x parameter

n EQU 2 ;defines n parameter

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

ENTRY

Main ADR sp,stack ;define the stack

ADD sp, #8 ;reserving two blocks for the x and n parameters

MOV r0, #x ;prepare the parameter x for the stack which is the base in x^n

MOV r1, #n ;prepare the parameter n which is the exponent in x^n

STR r0, [sp,#-8]! ;push the parameter x on the stack

STR r1, [sp,#-4]! ;push the parameter n on the stack

ADD sp, sp,#4 ;prepare an area in the stack for ther return value to be placed

BL Pow ;call the Pow subroutine to begin the recursive call

LDR r0, [sp,#-4] ;remove the value r0 from the stack and load it in register 0.

ADR r1, result ;retrieve the address of the variable that is named "result"

STR r0, [r1] ;store the contents of register one (which contains the adress of result) in register 0 to end the program.

Loop B Loop ;infinite loop - end of program.

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

AREA factorial, CODE, READONLY

Pow STMEA sp!, {r0-r2,fp,lr} ;push general registers, as well as fp and lr

MOV fp, sp ;set the fp for this call

Check LDR r0, [fp,#-32] ;retrieve the parameter holding the value x from accessing the frame pointer

LDR r1, [fp,#-28] ;retrieve the parameter holding the value n from accessing the frame pointer

CMP r1, #0 ;subtracts r1 - 0 to make the comparison if r1 = 0

BNE Comp ;if r1 is not equal to 0, branch to Comp

MOV r0, #1 ;otherwise, r1 (x) = 0, put 1 in register 0

STR r0, [fp,#-24] ;store the value of r0 in location pointed at by frame pointer with an offset of -24 (6 down)

B Return ;brnch to Return

Comp AND r2, r1, #1 ;seeing if n is odd by doing "if (n & 1)""

CMP r2, #1 ;if register is equal to one (r2 - 1)

BNE Even ;if they are not equal, then it is even and branch to Even.

Odd ADD sp, #8 ;the value in r2 is odd, so add 8 to the stack pointer

STR r0, [sp, #-8] ;store the value of parameter x in register 0 in location of stack pointer with offset -8 (4 spaces down)

SUB r1, #1 ;subtract register 1 by a value of 1 to decrement it

STR r1, [sp, #-4] ;store this value in register 1 in location of stack pointer to the space below register 0 (offset -4)

ADD sp, #4 ;create another space for return value

BL Pow ;branch to Pow to call the recursive function again

LDR r1, [sp, #-4] ;loads value in register 1 (parameter n) in location stack pointer - 4.

MUL r2, r0, r1 ;multiply the value of x by n in registers r0 and r1 and put the result in register 2.

STR r2, [fp, #-24] ;store the result of this computation in location frame pointer - 24.

B Return ;branch to Return to compute final result and end the program

Even ASR r1, #1 ;divide the value in register 1 by 2 using an arithmetic shift right.

ADD sp, #8 ;increase the stack pointer by 8

STR r1, [sp, #-4] ;store the value of register 1 in location stack pointer - 4

STR r0, [sp, #-8] ;stores the value os register 0 in stack pointer - 8 (below val of r1)

ADD sp, #4 ;create another space for return value

BL Pow ;branch to Pow to call the recursive function again

LDR r1, [sp, #-4] ;loads the value in register 1 (parameter n) in location stack pointer - 4

MUL r2, r1, r1 ;multiplies the value in register 1 by itself and places the result in register 2.

STR r2, [fp, #-24] ;store the value in register 2 in location of frame pointer - 24 (offset).

B Return ;branch to Return to compute final result and end the program

Return MOV sp,fp ;collapsing the current space by moving frame pointer back into stack pointer

LDMEA sp!,{r0,r1,r2,fp,pc} ;collapsing the registers by reloading them into stack pointer, substituting PC for LR.

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

AREA factorial, DATA, READWRITE

result DCD 0x00 ;final result

SPACE 0xB4 ;space for the stack

stack DCD 0x00 ;allocating memory for the stack

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

END

;-----

Structure of the Stack Frame

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| X = 6 |
| N = 2 |
| Old FP |
| R0 |
| R1 |
| R2 |
| Old Fp |
| lr |
|  |

-32

-28

-24

-20

-16

-12

-8 🡨 SP

-4

0

4

8

12

16 🡨 FP

20

24

|  |
| --- |
|  |
|  |

28

32

Q: How many stack frames are needed to calculate xn when n = 0…12?

Since each frame requires 32 bytes, frame x^0 = 32 bytes (1 frame), x^1 = 64 (2 frames), x^2 = 96 (3 frames), x^3 = 128 (4 frames), x^4 = 160 (5 frames).

Keep increasing 32 per stack frame, x^12 = 416 bytes with 13 frames.