[16 marks] 7. This is a question about writing ARM assembly-language code.

[5 marks] (a) Write a subroutine named `ROOT` in ARM assembly language that has an input i and produces an output $\sqrt{i}$. **Use this simple algorithm**: find the largest number, j, for which $j^2 \leq i$ (recall that ARM has a multiply instruction `MUL`). The value of *i* is passed to your subroutine in register `R0`, and you should also return the result, j, in register `R0`. Show your **Answer** in the space below.

Solution:

```
// Subroutine finds square root of R0, returns result in R0
ROOT:  MOV   R1, #0       // j = 0
WHILE: MUL   R2, R1, R1   // j^2
       CMP   R2, R0       // check exit condition
       BGE   DONEW        // exit if j^2 >= i
       ADD   R1, #1
       B     WHILE
DONEW: SUBGT R1, #1       // if (j^2 > i) --j
       MOV   R0, R1       // return result
DONE:  MOV   PC, LR
```

[2 marks] (b) Write a subroutine named `MOD` in ARM assembly language that produces the modulus, n `%` i. (The `%` is the *remainder* that is produced by the division operation $n \div i$). The values of n and i are passed to your subroutine in registers `R0` and `R1`, respectively. You should also return the result, $n \bmod i$, in register `R0`. Show your **Answer** in the space below.

Solution:

```
// Subroutine calculates the modulus R0 = R0 % R1
MOD:    MOV     R2, #0
MODL:   CMP     R0, R1
        BLT     ENDM
        SUB     R0, R1
        ADD     R2, #1
        B       MODL
ENDM:   MOV     PC, LR    // modulus is in R0
```

[9 marks] (c) For this part you are to write a main program in ARM assembly language that makes use of the subroutines created in parts (a) and (b), above. Your program has to determine if a given input value, N, is a *prime number*. If it is, you should store a 1 into a memory location `P`. If `N` is not prime, then you should store 0 into `P`.

You are to use the algorithm given below, which is provided in *pseudo-code*.

```
N = 17;        // N could be any integer
P = 0;         // P will be 1 if N is prime, else 0

if (N == 2)
    P = 1;     // 2 is prime
else if ((N & 1) == 1){  // only check odd values of N
    P = 1;     // until proven otherwise
    for (int i = 3; (i ≤ √N) && (P == 1); i = i + 2)
        if (N % i == 0)
            P = 0;
}
```

The special case of $n = 2$, which is the only *even* prime number, is handled at the beginning of the algorithm. Then, for the *odd* numbers the `for` loop checks whether N is divisible by any (odd) number, i, from 3 to $\sqrt{N}$. If a divisor (with remainder of 0) is found, then N is not prime and the loop terminates.

Write your ARM program that implements this algorithm in the space on the next page. A good approach for completing this task is to try to *translate* each line of the algorithm directly into assembly-language statements. To implement the operation $\sqrt{n}$ use your `ROOT` subroutine from part (a), and implement the modulus operator `%` with your `MOD` subroutine from part (b). Note that the & operator in the pseudo-code represents the logical `AND` *operator*, and && corresponds to a logical AND *condition*.

To make it easier to mark your answer, **you are required to to use some specific registers**, as shown on the next page. Use register `R4` to hold the value N, and read this value from memory. Also, use `R11` to hold the result P, which you should store into memory at the end of your code.

Solution:

```
_start: MOV     R4, #N
        LDR     R4, [R4]
        MOV     R11, #0     // result P will be in R10

        CMP     R4, #2      // if (N == 2)
        MOVEQ   R11, #1     // prime
        BEQ     EXIT
        ANDS    R0, R4, #1  // else check if odd
        BEQ     EXIT        // if even, skip
        MOV     R11, #1     // assume prime

        MOV     R5, #3      // i = 3
        MOV     R0, R4
        BL      ROOT
        MOV     R6, R0      // R6 = sqroot(n)
FOR:    CMP     R5, R6      // i <= sqroot(n) ?
        BGT     EXIT
        MOV     R0, R4      // check modulus
        MOV     R1, R5
        BL      MOD         // R0 = n % i
        CMP     R0, #0      // if (N % i) == 0)
        MOVEQ   R11, #0     // not prime
        BEQ     EXIT
        ADD     R5, #2      // i += 2
        B       FOR

EXIT:   MOV     R0, #P
        STR     R11, [R0]

END:    B       END

N:      .word   17
P:      .word   0
```