❖ MES PROGRAMS(IA-2)

1. Construct/Develop an ALP to find the sum of N integers stored in an array. The result is stored in internal RAM.

```
        AREA ADDITION, CODE, READONLY
ENTRY
        MOV R5, #5      ; initialize counter(n) to 5
        MOV R0, #0      ; initialize sum to 0
        LDR R1, =VALUE1   ; loads address of first value from VALUE1
LOOP:  LDRH R3, [R1], #02    ; loads the contents of r1 to r3 and increments by 2
        ADD R0, R0, R3      ; add r0 and r3 and move result to r0
        SUBS R5, R5, #01    ; decrement counter by 1 till 0
        BNE LOOP    ;       branch back to loop label
        LDR R4, =RESULT   ; loads the address of result
        STR R0, [R4]       ; stores result in contents of r4
BACK B BACK
VALUE1 DCW 0x0001,0x0002,0x0003,0x0004,0x0005   ; array of 16-bit numbers
        AREA DATA2, DATA, READWRITE    ; to store result in given address
RESULT DCD 0x0
        END
```

2. Construct/Develop an ALP to find the smallest number in an array.

```
        AREA SMALLEST, CODE, READONLY
ENTRY
        MOV R5, #3          ; initialize the counter(n) to 3
        LDR R1, = VALUE1 ; loads address of first value from VALUE1
        LDR R2, [R1], #4    ; loads content of r1 to r2 and increments by 4
LOOP: LDR R4, [R1], #4     ; loads content of second address to r4 and increments by 4
        CMP R2, R4          ; compares r2 and r4
        BLS LOOP2           ; branches to loop2 label if r2<r4
        MOV R2, R4          ; moves r4 value to r2 otherwise
```

```
LOOP2: SUBS R5, R5, #1        ; decrement counter by 1
        BNE LOOP              ; branch back to loop label
        LDR R4, = RESULT      ; loads the address of result
        STR R2, [R4]          ; stores the result in contents of r4
BACK B BACK
VALUE1 DCD 0x0001,0x0002,0x0003,0x0004     ; address of 32-bit numbers
        AREA DATA2, DATA, READWRITE        ; to store result in address
RESULT DCD 0x0
        END
```

[NOTE: CONSIDER 'N-1' VALUE WHILE INITIALIZING THE COUNTER]

3. Construct/Develop an ALP to find the largest number in an array.

```
        AREA SMALLEST, CODE, READONLY
ENTRY
        MOV R5, #3            ; initialize the counter(n) to 3
        LDR R1, = VALUE1      ; loads address of first value from VALUE1
        LDR R2, [R1], #4      ; loads content of r1 to r2 and increments by 4
LOOP: LDR R4, [R1], #4        ; loads content of second address to r4 and increments by 4
        CMP R2, R4            ; compares r2 and r4
        BGT LOOP2             ; branches to loop2 label if r2>r4
        MOV R2, R4            ; moves r4 value to r2 otherwise
LOOP2: SUBS R5, R5, #1        ; decrement counter by 1
        BNE LOOP              ; branch back to loop label
        LDR R4, = RESULT      ; loads the address of result
        STR R2, [R4]          ; stores the result in contents of r4
BACK B BACK
VALUE1 DCD 0x0001,0x0002,0x0003,0x0004     ; address of 32-bit numbers
        AREA DATA2, DATA, READWRITE        ; to store result in address
RESULT DCD 0x0
        END
```

4. Construct/Develop an ALP to sort the array in ascending order.

```
        AREA ASCENDING, CODE, READONLY
ENTRY
        MOV R8, #4          ; initialize counter to 4
        LDR R2, =CVALUE     ; load address of CODE region
        LDR R3, =DVALUE     ; load address of DATA region
LOOP
        LDR R1, [R2], #4    ; load values from code region
        STR R1, [R3], #4    ; store values to data region
        SUBS R8, R8, #1     ; decrement the counter
        BNE LOOP            ;  loop back to loop0 label
START1
        MOV R5, #3          ; initialize counter to 3 (n-1 for Bubble Sort)
        MOV R7, #0          ; flag to denote if an exchange has occurred
        LDR R1, =DVALUE     ; load the address of the first value
LOOP1
        LDR R2, [R1], #4    ; load the first array element
        LDR R3, [R1]        ; load the second array element
        CMP R2, R3          ; compare r2 and r3
        BLT LOOP2           ;  if r2<r3, go to loop2

   ; Interchange r2 and r3 if r2 > r3
        STR R2, [R1], #-4   ; store r2 back in its original position
        STR R3, [R1]        ; store r3 in the next position
        MOV R7, #1          ; set flag indicating an exchange has taken place
        ADD R1, #4          ;  restore the pointer address(r1)
LOOP2
        SUBS R5, R5, #1     ; decrement counter
        BNE LOOP1           ; loop back to loop1
```

```
        CMP R7, #0          ; compare the exchange flag
        BNE START1          ; if flag is not zero, go to START1 for another pass
BACK B BACK
CVALUE DCD 0x0001, 0x0002, 0x0003, 0x0004  ; Array of 32-bit numbers in code region

        AREA DATA1, DATA, READWRITE   ; Array of 32-bit numbers in data region
DVALUE DCD 0x0, 0x0, 0x0, 0x0
        END
```

5. Construct/Develop an ALP to sort the array in descending order.

```
        AREA DESCENDING, CODE, READONLY
ENTRY
        MOV R8, #4          ; initialize counter to 4
        LDR R2, =CVALUE     ; load address of CODE region
        LDR R3, =DVALUE     ; load address of DATA region
LOOP
        LDR R1, [R2], #4    ; load values from code region
        STR R1, [R3], #4    ; store values to data region
        SUBS R8, R8, #1     ; decrement the counter
        BNE LOOP            ;  loop back to loop0 label
START1
        MOV R5, #3          ; initialize counter to 3 (n-1 for Bubble Sort)
        MOV R7, #0          ; flag to denote if an exchange has occurred
        LDR R1, =DVALUE     ; load the address of the first value
LOOP1
        LDR R2, [R1], #4    ; load the first array element
        LDR R3, [R1]        ; load the second array element
        CMP R2, R3          ; compare r2 and r3
        BGT LOOP2           ;  if r2>r3, go to loop2
```

```
; Interchange r2 and r3 if r2 < r3
        STR R2, [R1], #-4   ; store r2 back in its original position
        STR R3, [R1]        ; store r3 in the next position
        MOV R7, #1          ; set flag indicating an exchange has taken place
        ADD R1, #4          ; restore the pointer address(r1)
LOOP2
        SUBS R5, R5, #1     ; decrement counter
        BNE LOOP1           ; loop back to loop1
        CMP R7, #0          ; compare the exchange flag
        BNE START1          ; if flag is not zero, go to START1 for another pass
BACK B BACK
CVALUE DCD 0x0001, 0x0002, 0x0003, 0x0004    ; Array of 32-bit numbers in code region

        AREA DATA1, DATA, READWRITE   ; Array of 32-bit numbers in data region
DVALUE DCD 0x0, 0x0, 0x0, 0x0
        END
```

6. Construct/Develop an ALP to count the number of ones and zeroes in a given number.

```
        AREA ONEZERO, CODE, READONLY
ENTRY
        MOV R2, #0          ; initialize counter for ones
        MOV R3, #0          ; initialize counter for zero
        MOV R6, #0x0001     ; loads address of given number
LOOP
        MOV R1, #32         ; initialize bit counter to 32
        LDR R0, [R6]    ; loads the content of r6 to r0
LOOP0
        MOVS R0, R0, ROR #1  ; rotate right and place the bit in the carry flag
        BHI ONES                ; if carry is set (bit was 1), branch to ones
```

```
ZEROS
        ADD R3, R3, #1      ; increment zero counter
        B LOOP1             ; skip the ones increment
ONES
        ADD R2, R2, #1      ; increment one counter
LOOP1
        SUBS R1, R1, #1     ; decrement bit counter
        BNE LOOP0           ; if bit counter is not zero, repeat
BACK B BACK
        END
```

7. Construct/Develop an ALP to move a block of data from source to destination locations.

```
        AREA BLOCK, CODE, READONLY
ENTRY
        LDR R1, = SRC    ; loads address of SRC region
        LDR R2, = DST    ; loads address of DST region
        MOV R3, #5       ; initialize counter to 5
LOOP    MOV R4, [R1], #4  ; move contents of r1 to r4 and increment by 4
        STR R4, [R2], #4     ; store r4 value in contents of r2 and increment by 4
        SUBS R3, R3, #1      ; decrement counter by 1
        BNE LOOP             ; branch to loop label
BACK B BACK
SRC DCD 0x0001, 0x0002, 0x0003, 0x0004, 0x0005       ;  array of 32-bit numbers
        AREA DATA2, DATA, READWRITE          ;  address of result array
DST DCD 0x0
        END
```

8. Construct/Develop an ALP to exchange a block of data of source1 and source2 locations

```
                AREA EXCHANGE, CODE, READONLY
ENTRY
        LDR R1, = SRC1    ; loads address of SRC1 region
        LDR R2, = SRC2    ; loads address of SRC2 region
        MOV R3, #5        ; initialize counter to 5
LOOP    MOV R4, [R1]  ; move contents of r1 to r4
        MOV R5, [R2]  ; move contents of r5 to r2
        STR R4, [R2], #4     ; store r4 value in contents of r2 and increment by 4
        STR R5, [R1], #4     ; store r5 value in contents of r1 and increment by 4
        ADD R4, R4, #4          ; increment r4 and r5 by 4
        ADD R5, R5, #4
        SUBS R3, R3, #1      ; decrement counter by 1
        BNE LOOP             ; branch back to loop label
BACK B BACK
SRC1 DCD 0x0001, 0x0002, 0x0003, 0x0004, 0x0005       ;  arrays of 32-bit numbers
SRC2 DCD 0x000A, 0x000B, 0x000C0x000D, 0x000E
```

9. Construct/Develop an ALP to check whether the given number is even or odd number

```
        AREA EVENODD, CODE, READONLY
ENTRY
        MOVE R1, #3                 ; move the given number to r1
        ROR R1, R1, #1              ; rotate right the number by 1
        BHI LOOP                    ; branch to loop label if carry is high(1)
        MOVE R0, #0                 ; set r0 to 0
LOOP  MOVE R0, #1                   ;  set r0 to 1
BACK B BACK
        END
```

[NOTE: IF R0= 0, NUMBER IS EVEN; R0=1, NUMBER IS ODD]

10. Construct/Develop an ALP to find the GCD of 2 numbers

```
        AREA GCD, CODE, READONLY
ENTRY
        MOV R1, #48      ;  move first number to r1
        MOV R2, #18      ; move second number to r2
LOOP    CMP R1, R2       ; compare r1 and r2
        BEQ GCD          ; if r1==r2 , branch to gcd label
        BHI LOOP2        ; if r1>r2,  branch to loop2 label
        SUB R2, R2, R1   ; if r1<r2, subtract r1 from r2 and move result to r2
        B LOOP           ; branch back to loop label
LOOP2   SUB R1, R1, R2       ; subtract r2 from r1 and move result to r1
        B LOOP               ; branch back to loop label
GCD     MOV R0, R1           ; move the gcd result from r1 to r0
BACK B BACK
        END
```

11. Write a C code and it's corresponding assembly code to print the squares of the integer 0 to 9.

```
        AREA SQUARE, CODE, READONLY
ENTRY
        MOV R0, #0       ; move 0 to r0
LOOP  LDR R1, = DEST     ; loads address of first value from DEST
        CMP R0, #10      ; compare r0 value with 10
        BEQ BACK         ; if equal, branch to back label
        MUL R2, R0, R0   ; square r0 and move to r2 otherwise
        STR R2, [R1], #4 ; store the result from r1 to contents of r1 and increment by 4
        ADD R0, R0, #1       ; increment r0 by 1
        B LOOP           ; branch back to loop
BACK  B BACK
        AREA DATA2, DATA, READWRITE      ; result array address
DEST 0x0      END
```

C Program:

```
#include <stdio.h>
int square(int i);
int main(void)
{
        for (int i=0; i<10; i++)
                printf("Square of %d is %d\n", i, square(i));
}
int square(int i){
        return i*i;  }
```

12. With an example show how to call a subroutine from an assembly routine.(Example of printf as a subroutine from C Libraries)

```
        AREA SUBRCALL, CODE, READONLY
 EXPORT main
 IMPORT |Lib$$Request$$armlib|, WEAK    ; import lib, request, armlib libraries
 IMPORT main          ; C library entry
 IMPORT printf        ; prints to stdout
 i RN 4               ; define i as register 4
; int main(void)
main    STMFD sp!, {i, lr}     ; stores i and lr into the stack pointed to by sp
         MOV i, #0              ; initialize i to 0
loop    ADR r0, print_string    ; load address of print_string to r0
        MOV r1, i               ; move i value to r1
        MUL r2, i, i            ; square i value and store in r2
        BL printf               ; calls printf to print the message
        ADD i, i, #1            ; increments i by 1
        CMP i, #10              ; compare i with 10
        BLT loop                ; if i<10, branch to loop label
        LDMFD sp!, {i, pc}      ; restores registers 'i' and pc to return from subroutine
 print_string
     DCB "Square of %d is %d\n", 0  ; define the string to print message terminated with '\0'
  END
```

13. Write the sumof() program in assembly routine

```
AREA SUMOF, CODE, READONLY
EXPORT sumof
N RN 0  ; N as R0(number of elements to sum)
 sum RN 1 ; sum as R1(to store current sum)
; int sumof(int N, ...)
sumof  SUBS N, N, #1 ; decrement N by 1 (check if we have one element)
          MOVLT sum, #0 ; if N is less than 0 (no elements), set sum to 0
          SUBS N, N, #1 ;  decrement N by 1 (check if we have two elements)
          ADDGE sum, sum, r2 ; if N >= 0, add r2 value to sum
          SUBS N, N, #1 ; decrement N by 1 (check if we have three elements)
          ADDGE sum, sum, r3 ; if N >= 0, add r3 value to sum
           MOV r2, sp          ; load the stack pointer into r2 (top of stack)
loop
          SUBS N, N, #1 ; decrement N by 1 (check if we have another element)
          LDMGEFD r2!, {r3} ; Load a word from stack into r3 (for each additional element)
          ADDGE sum, sum, r3   ; if N >= 0, add r3 to sum
          BGE loop                    ; branch back to loop label if N>=0
          MOV r0, sum                 ; move sum value to r0
           MOV pc, lr                 ; return to calling function
 END
```