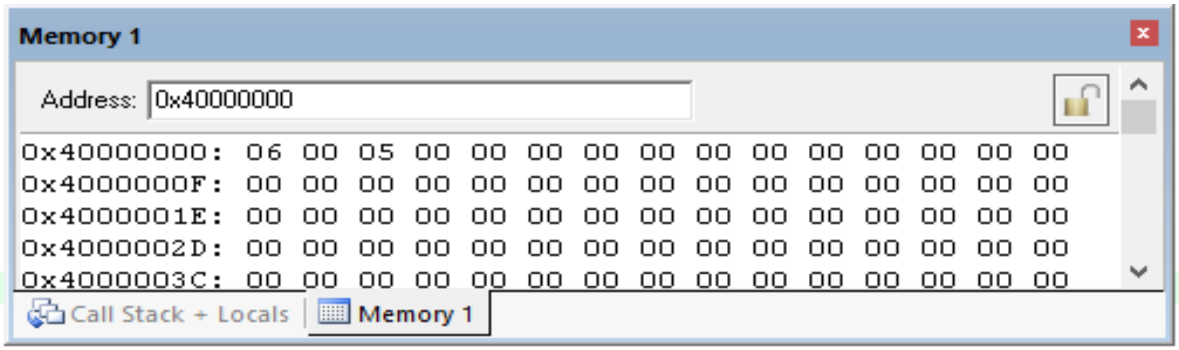
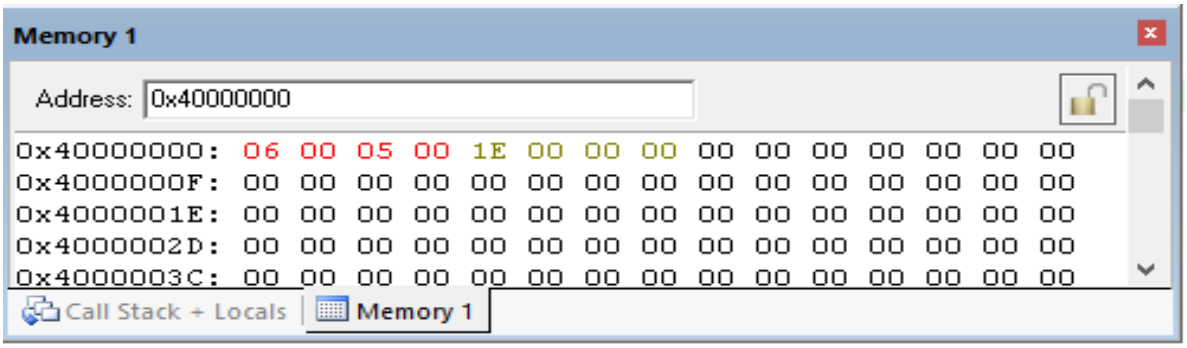
1. **Write a program to multiply two 16 bit Binary numbers.**

```
1
2   AREA Program, CODE, READONLY
3     ENTRY
4
5     LDR     R0, MEMORY        ; load Address of memory
6     LDRH    R1, [R0]          ; load First number
7     LDRH    R2, [R0,#2]       ; load Second number
8     MUL     R3, R1, R2        ; R2 = R1 x R2
9     STR     R3, [R0,#4]       ; Store the result
10 B1   B   B1
11
12
13 MEMORY DCD 0x40000000      ; Address of First 16 bit number
14
15  END
```

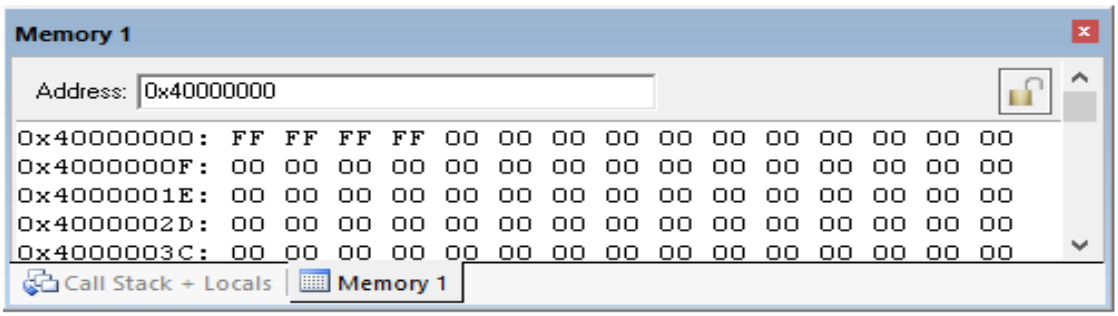
Before Execution:



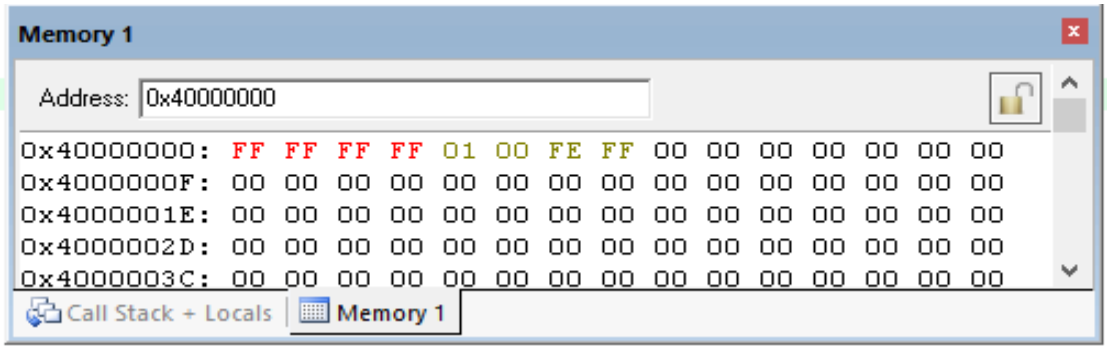
After Execution:



Before Execution:



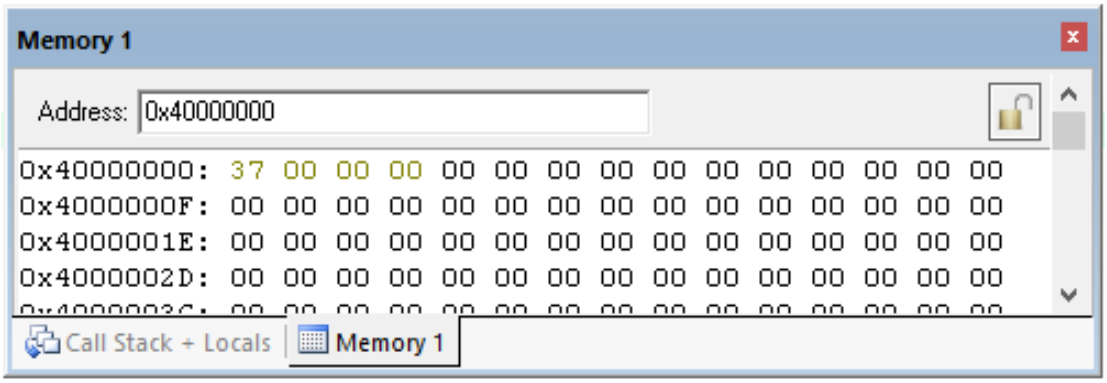
After Execution:



2. Write a program to find the sum of First 10 integer numbers.

After Execution:

$1+2+3+4+5+6+7+8+9+10= (55)_{10}= (37)_{16}$

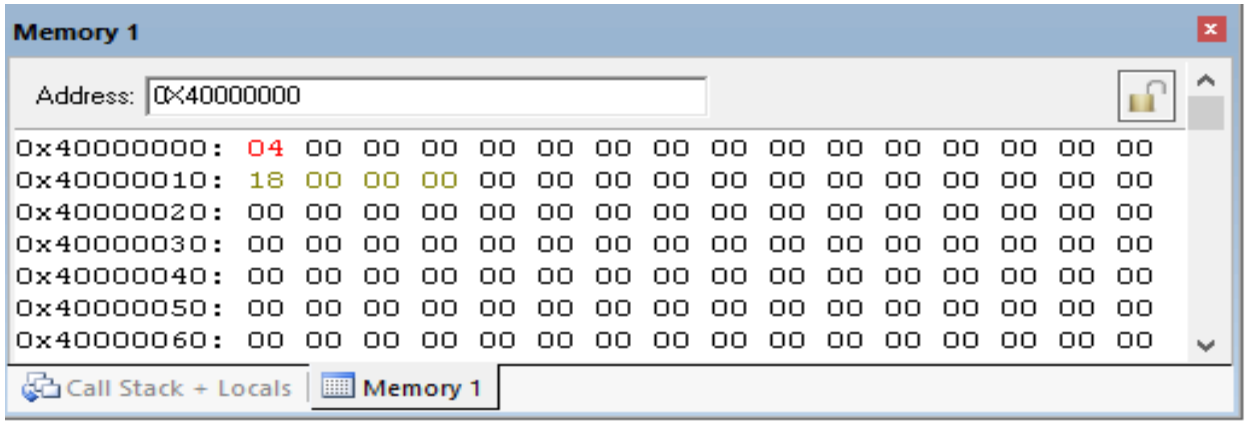


```
1
2   AREA    SUM,CODE,READONLY
3       ENTRY
4
5       MOV     R0,#10          ;Counter (10 integer number
6       MOV     R1,#0           ;Partial sum
7       MOV     R2,#1           ;First number
8
9 NEXT  ADD      R1,R1,R2       ;add partial sum with first number
10      ADD      R2,#1          ;update next integer number
11      SUBS     R0,#1          ;Decrement the counter
12      BNE       NEXT          ;if counter != 0 then repeat
13      LDR       R3,=RES       ;Get the address of the result
14      STR       R1,[R3]       ;store the result( Final sum)
15 B1   B           B1          ;stop
16
17
18    AREA  Data1,DATA,READWRITE
19
20 RES   DCD  0
21      END
```

3. Write a Program to find factorial of a number.

After Execution:

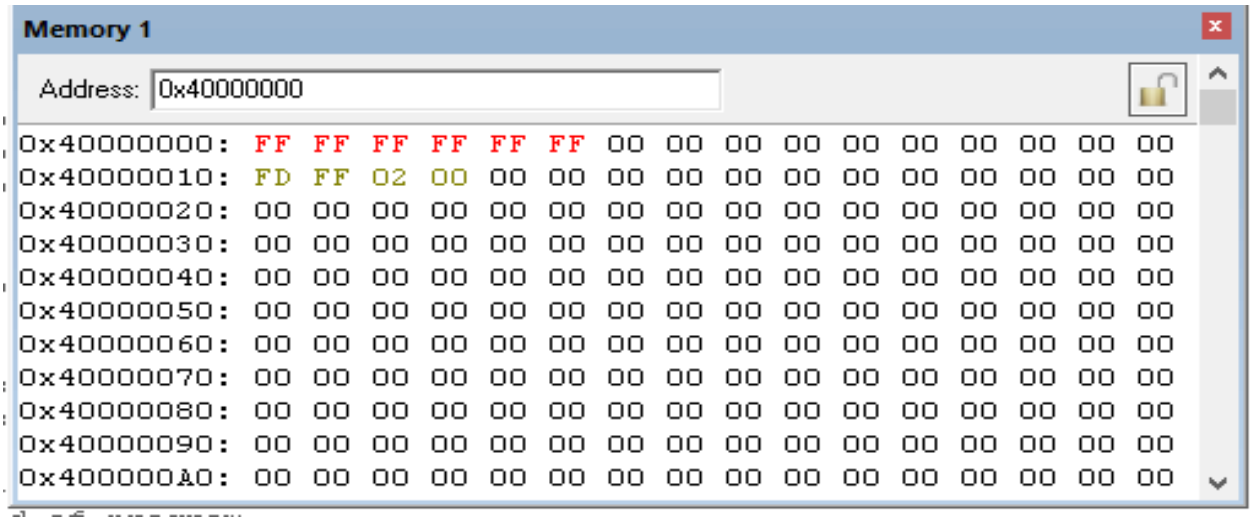
4! = 4X3X2X1 = $(24)_{10}$ = $(18)_{16}$



```
1
2   AREA    Factorial, CODE, READONLY
3         ENTRY
4         LDR       R0,MEMORY    ;get the address of the memory
5         LDRB      R1,[R0]      ;Read input
6         MOV       R2,#1        ;Initial result( 0! =1)
7         CMP       R1,#0        ; Check for 0
8         BEQ       STORE        ; if input is 0 then store 1 as result
9         MOV       R2,R1        ; store input in R2
10 UP     ADD       R1,R1,#-1    ; decrement input by 1 and store it in R1
11        CMP       R1,#0        ; check for 0
12        BEQ       STORE        ; if it is 0 then store the result
13        MUL       R3,R2,R1     ;R2=n, R1=n-1 , R3 <-- R2 X R1
14        MOV       R2,R3        ; Store Partial product in R2
15        B         UP           ; Repeat
16 STORE  LDR       R0,RESULT
17        STR       R2,[R0]      ; Store the result.
18 HERE   B         HERE
19
20 MEMORY DCD      0X40000000   ; Memory location for Input
21 RESULT DCD      0X40000010   ; Memory location for Output
22        END
```

4. Write a program to add an array of 16 bit numbers and store the 32 bit result in the memory

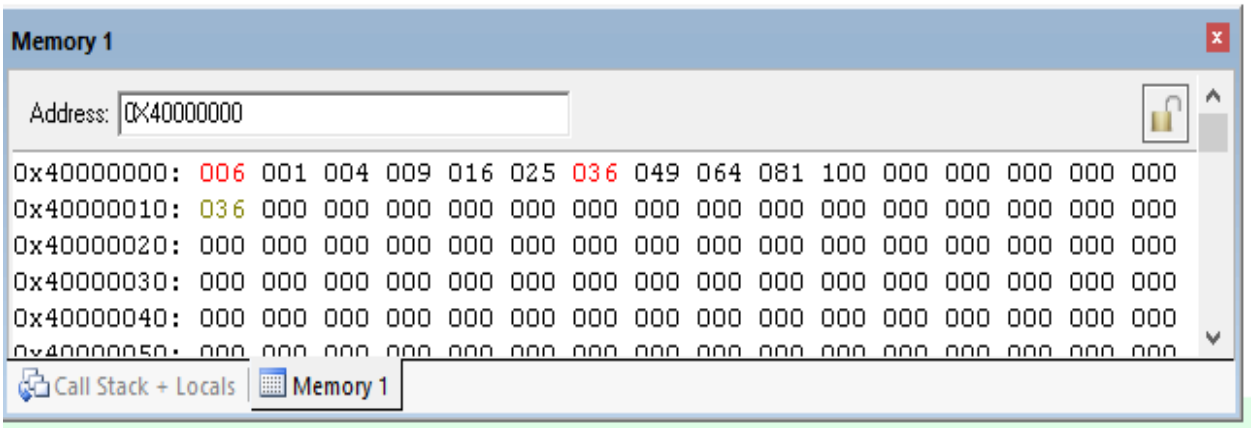
After Execution:



```
1
2  AREA  SUM, CODE, READONLY
3          ENTRY
4          LDR       R0, MEMORY      ; Load Starting address of the array
5          MOV       R1,#3           ; load array Size(Counter)
6          LDRH      R2,[R0]         ; load First number
7          ADD       R1,#-1          ; Decrement Counter
8  UP      ADD       R0,R0,#2        ; Increment Ptr by 2
9          LDRH      R3,[R0]         ; load Second  number
10         ADD       R2, R3, R2      ; R2 <-- R3+R2
11 NEXT    ADD       R1,#-1          ; Decrement Counter
12         CMP       R1,#0           ; Is Cntr=0 ?
13         BNE       UP              ; if cntr != 0  Then Repeat
14         LDR       R0,RESULT
15         STR       R2,[R0]         ;  Store the result
16 Here    B         Here
17
18
19 MEMORY  DCD       0X40000000      ; Starting address of the Input data
20 RESULT  DCD       0X40000010      ; Starting address of the Output
21         END
```

5. Write a program to find the square of a number(1 to 10) Using look-up table.

After Execution:



```
1
2   AREA    SQR, CODE, READONLY
3           ENTRY
4           LDR     R0,LOOKUP    ;get the starting address of the Lookup table
5           LDR     R2,RESULT    ;get the starting address of the result
6           LDRB    R1,[R0]      ;Read the input
7           ADD     R0,R0,R1     ;calculate the address of the output(Square of the input)
8           LDRB    R3,[R0]      ;Read the square of the number
9           STRB    R3,[R2]      ;Store the result
10 HERE     B       HERE         ;stop
11
12 LOOKUP   DCD         0X40000000
13 RESULT   DCD         0X40000010
14          END
```

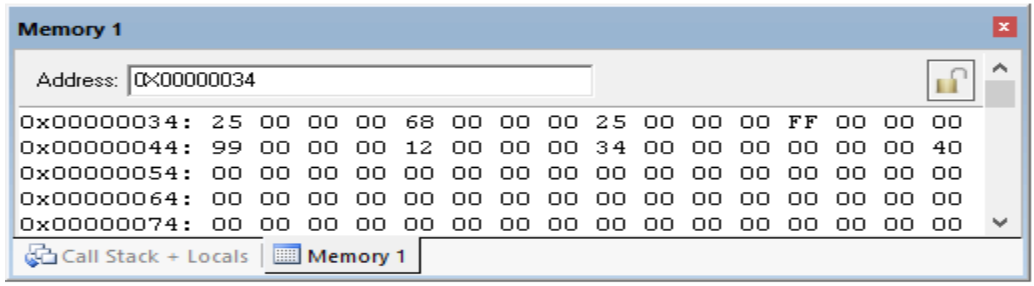
6. Write a program to find the largest/smallest number in an array of 32 bit numbers.

Code	Suffix	Flags	Meaning
0000	EQ	Z set	equal
0001	NE	Z clear	not equal
0010	CS	C set	unsigned higher or same
0011	CC	C clear	unsigned lower
0100	MI	N set	negative
0101	PL	N clear	positive or zero
0110	VS	V set	overflow
0111	VC	V clear	no overflow
1000	HI	C set and Z clear	unsigned higher
1001	LS	C clear or Z set	unsigned lower or same
1010	GE	N equals V	greater or equal
1011	LT	N not equal to V	less than
1100	GT	Z clear AND (N equals V)	greater than
1101	LE	Z set OR (N not equal to V)	less than or equal
1110	AL	(ignored)	always

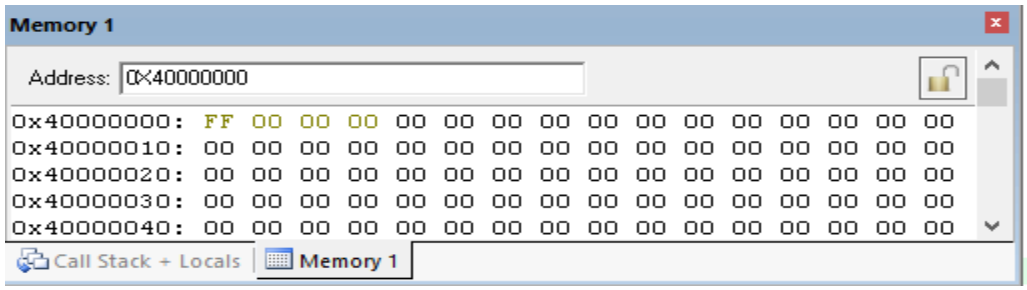
Table 4-2: Condition code summary

```
 1
 2   AREA      LARGEST,CODE,READONLY
 3             ENTRY
 4             MOV       R5,#5    ; Number of comparison(N-1 Comparison)
 5             LDR       R0,=A    ; get the starting address of the array
 6             LDR       R1,=RES  ; get the address of the result
 7             LDR       R2,[R0] ;read first number
 8 NEXT        ADD       R0,#4    ;update the pointer by 4
 9             LDR       R3,[R0] ;read second number
10             CMP       R2,R3    ;compare first number(R2) with second number(R3)
11             BHS       LARGE    ;(Unsigned  Higher or Same-HS)
12                                 ; this condn is true when R2 > R3
13             MOV       R2,R3
14 LARGE       SUBS      R5,#1      ;decrement comparison counter
15             BNE       NEXT
16             STR       R2,[R1]  ;store the result
17 B1          B         B1
18 A           DCD   0X25,0X68,0X25,0XFF,0X99,0X12
19
20     AREA  Data1,DATA,READWRITE
21 RES         DCD     0
22             END
23 ; NOTE: Replace BHS by BLS (Unsigned  Lower   or Same)
24 ; to find smallest number
```

Input:



OUTPUT:

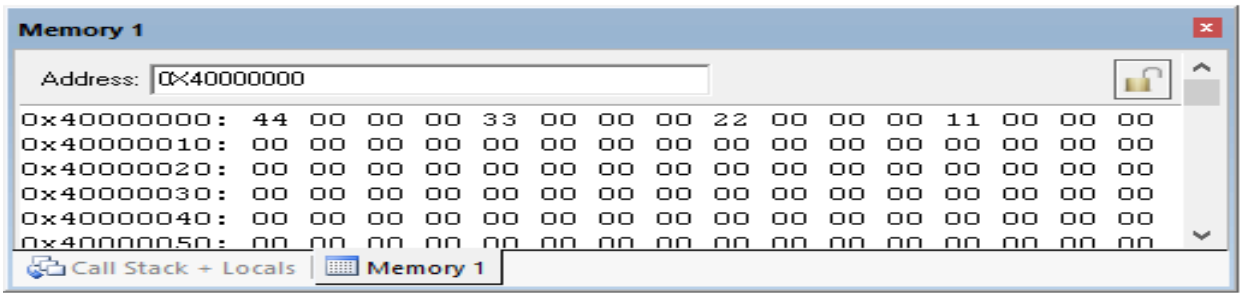


7. Write a program to arrange a series of 32 bit numbers in ascending/descending order.

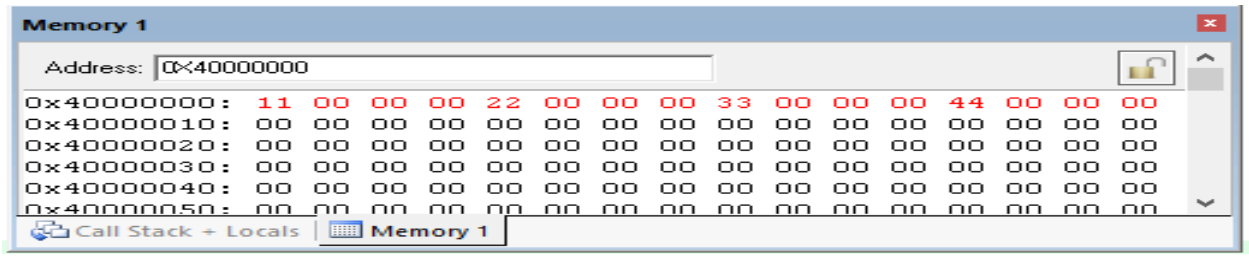
```
1
2       AREA  SORT, CODE , READONLY
3             ENTRY
4             MOV     R5,#3          ; N-1 Passes
5 NXTPASS     LDR     R0,A
6             MOV     R4,R5          ;N-1 Comparison
7 NXTCOMP     LDR     R2,[R0]        ;Read First number
8             MOV     R1,R2          ; Save it in R1
9             ADD     R0,#4          ; Update the pointer
10            LDR     R2,[R0]        ; Read Second number
11            CMP     R1,R2          ; Compare 1st  number with 2nd number
12            BLS     NOEXG          ; if  1st num < 2nd num then exchange is not required
13            STR     R1,[R0], #-4   ; Exchange the number
14            STR     R2,[R0], #4
15 NOEXG      SUBS    R4, #1         ; Decrement number of comparison
16            BNE     NXTCOMP
17            SUBS    R5, #1         ; Decrement number of Passes
18            BNE     NXTPASS
19 B1         B       B1
20 A          DCD     0X40000000
21            END
```

Note: Replace **BLS** by **BHS** for descending order

Before Execution

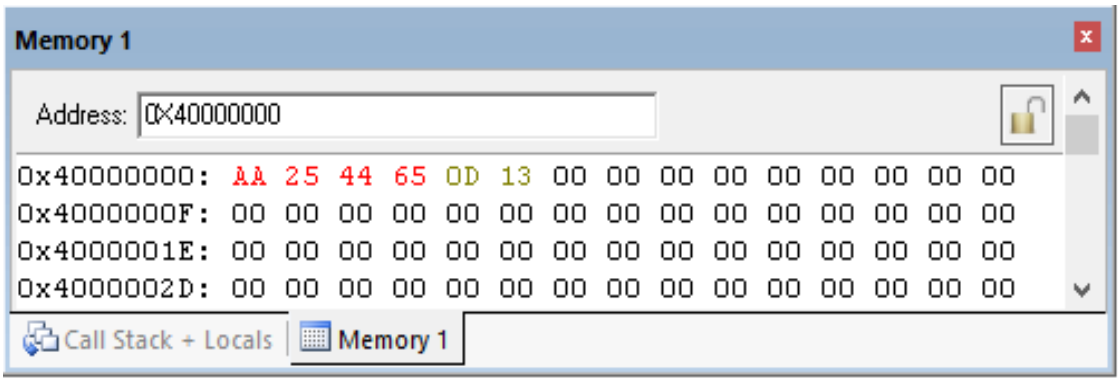


After Execution



8. Write a program to count the number of ones and zeros and place the result in two consecutive memory locations.

```
1
2     AREA Program, CODE, READONLY
3           ENTRY
4           LDR     R0,MEMORY    ; load the address of the Memory
5           LDR     R1,[R0]      ; load 32 bit number
6           MOV     R4,#32       ; load rotation count
7  ROTATE   RORS    R1,#1        ; rotate right by one bit, update cpsr
8           BCS     ONES         ; is carry is = 1
9           ADD     R3,R3,#1     ; increment zero's counter
10          B       NEXT         ; branch to next rotation
11 ONES     ADD     R2,R2,#1     ; increment one's counter
12 NEXT     ADD     R4,R4,#-1    ; Decrement the rotation count
13          CMP     R4,#0        ; is rotation count is zero
14          BNE     ROTATE       ;if  no, go to rotate
15          ADD     R0,R0,#04    ; load address of memory for no of one's
16          STRB    R2,[R0]      ; store no of one's
17          ADD     R0,R0,#1     ; load address of memory for no of zero's
18          STRB    R3,[R0]      ; store no of zero's
19 HERE     B       HERE
20
21 MEMORY   DCD     0X40000000   ; memory address
22          END
```



Input: 654425AA =
0110 0101 0100 0100 0010 0101 1010 1010

# of ones = $(13)_{10}$ = $(0D)_{16}$
# of zeros = $(19)_{10}$ = $(13)_{16}$