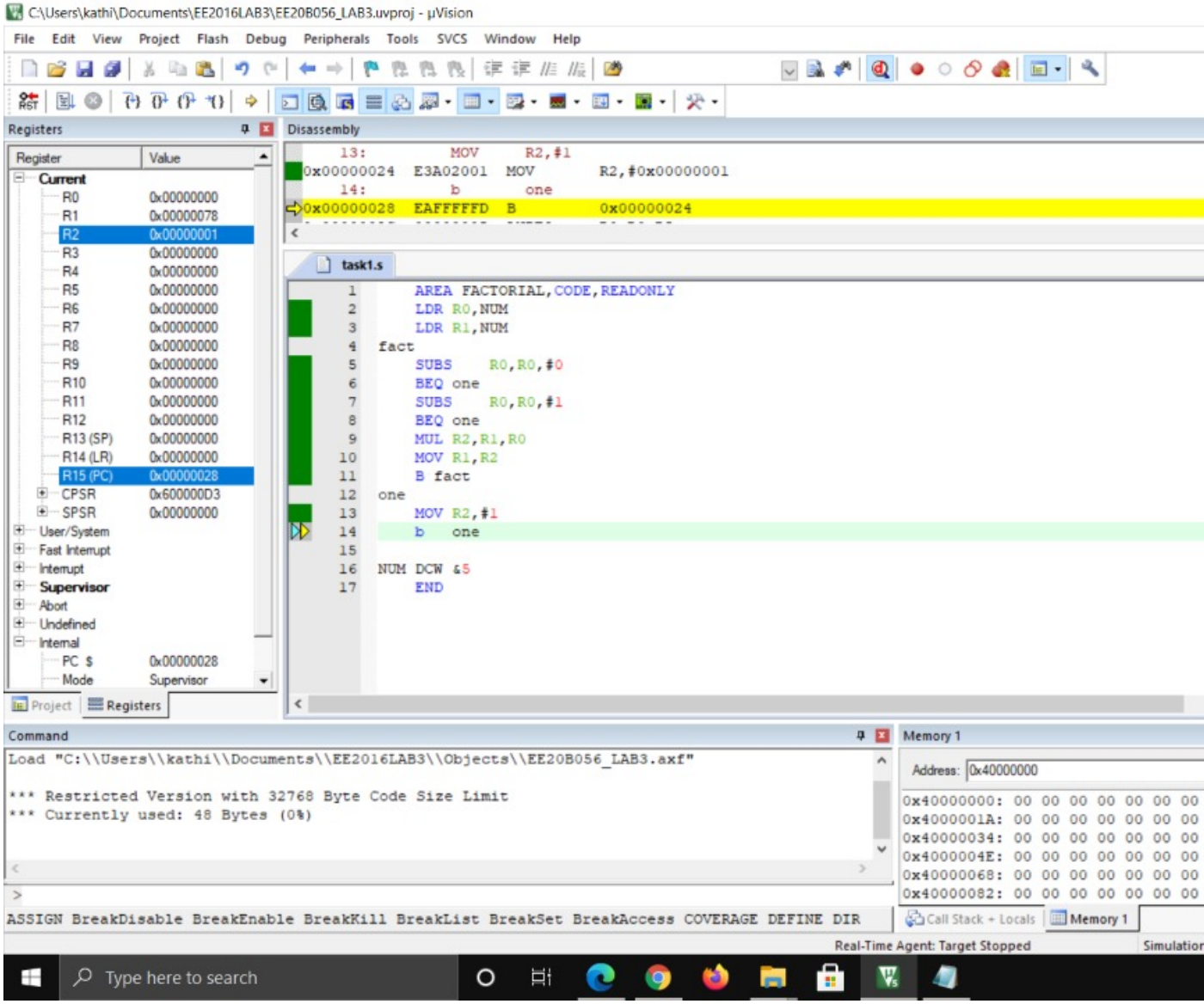
EE2016 LAB EXPERIMENT 3 – ARM BASICS

EE20B056

TASK 1 – FACTORIAL:

Compute the factorial of a given number using ARM processor through assembly programming



CODE:

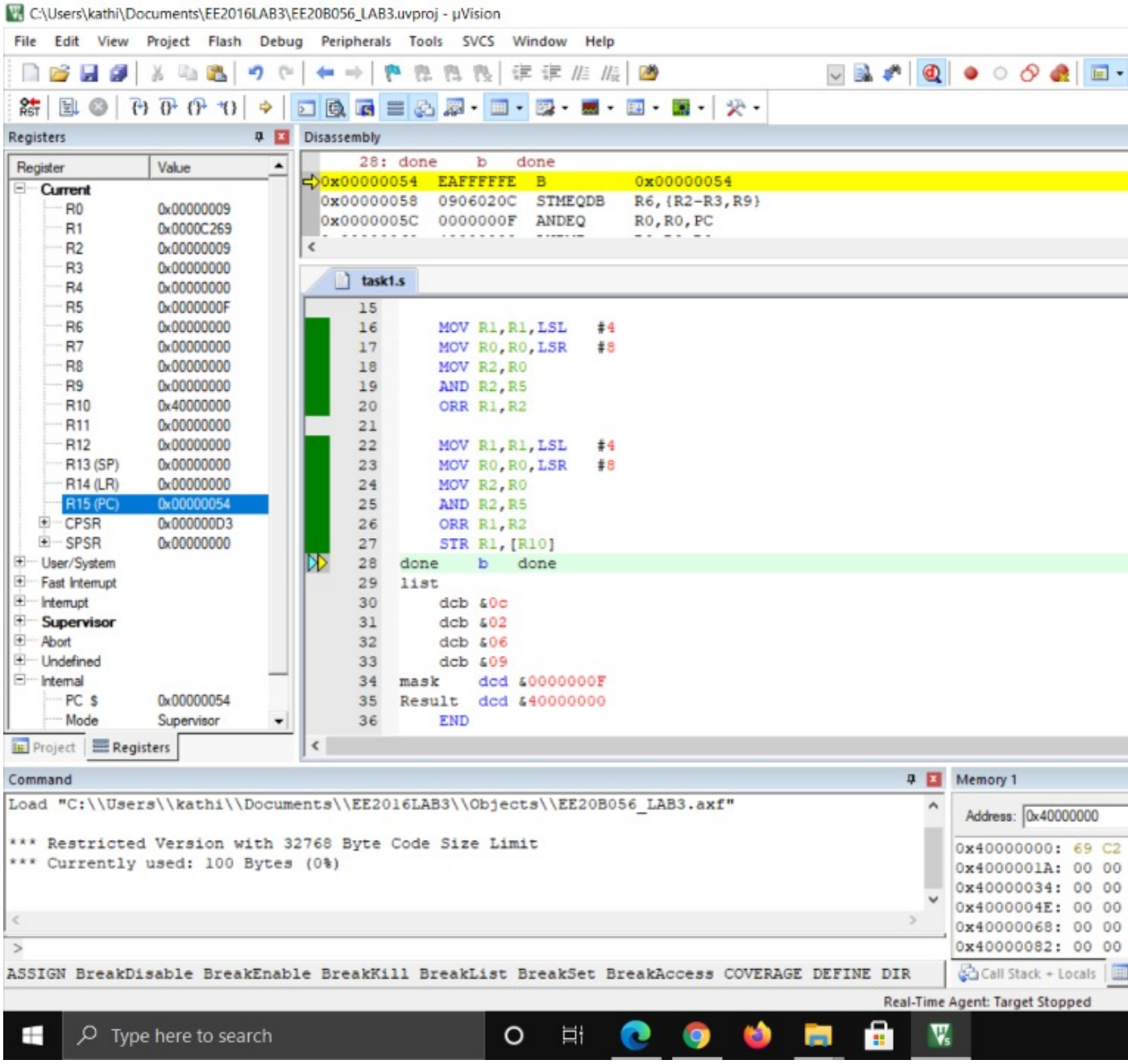
```
        AREA FACTORIAL,CODE,READONLY
        LDR     R0,NUM
        LDR     R1,NUM
fact
        SUBS    R0,R0,#0
        BEQ     one
        SUBS    R0,R0,#1
        BEQ     one
        MUL     R2,R1,R0
        MOV     R1,R2
        B fact
one
        MOV     R2,#1
        b       one

NUM     DCW     &5
        END
```

LOGIC: Given number is put in R0 and R1, firstly we check if given number is 1 if true, we jump to 'one' label meaning the factorial is 1, if given number is not 1 , we subtract the value in R0 by 1 and multiply with the value in R1 and put it in R2, and we copy this value back to R1 and proceed the same way. For a given value the process is shown below

[R0,R1,R2] : [5,-,-] → [5,5,-] → [4,5,-] → [4,5,20] → [4,20,20] → [3,20,20] → [3,20,60] → [3,60,60] → [2,60,60] → [2,60,120] → [2,120,120] → [1,120,120] → [1,120,1]

TASK 2 – 16-bit HALFWORD:

Combine the low four bits of each of the four consecutive bytes beginning at LIST into one 16-bit halfword. The value at LIST goes into the most signi cant nibble of the result. Store the result in the 32-bit variable RESULT.



CODE:

```
        AREA Program ,CODE,READONLY


        LDR R0, list ; four consecutive bytes as defined below
        LDR     R5,     mask ; 4 LSB bits alone are kept as 1s to be used as mask
```

```
        LDR     R10, Result ; the address being labelled as result is copied to this register

        MOV     R1,R0
        AND     R1,R5 ; the first 4 lsb bits alone are kept

        MOV     R1,R1,LSL       #4 ; the four bits are shifted to left,
        MOV     R0,R0,LSR       #8 ; eight bits here are shifted to the right so that the  MSB half byte is removed
        MOV     R2,R0
        AND     R2,R5 – mask the other bits than the lower half byte of the next byte alone
        ORR     R1,R2

        MOV     R1,R1,LSL       #4
        MOV     R0,R0,LSR       #8
        MOV     R2,R0
        AND     R2,R5
        ORR     R1,R2

        MOV     R1,R1,LSL       #4
        MOV     R0,R0,LSR       #8
        MOV     R2,R0
        AND     R2,R5
        ORR     R1,R2
        STR     R1,[R10]
done    b       done
list
        dcb     &0c
        dcb     &02
        dcb     &06
        dcb     &09
mask    dcd     &0000000F
```

```
Result  dcd     &40000000
        END
```

LOGIC:

R0 – LIST – 4 bytes sequenced as a single 32 bit

R1 – output - LSBs of the given 4 bytes in list

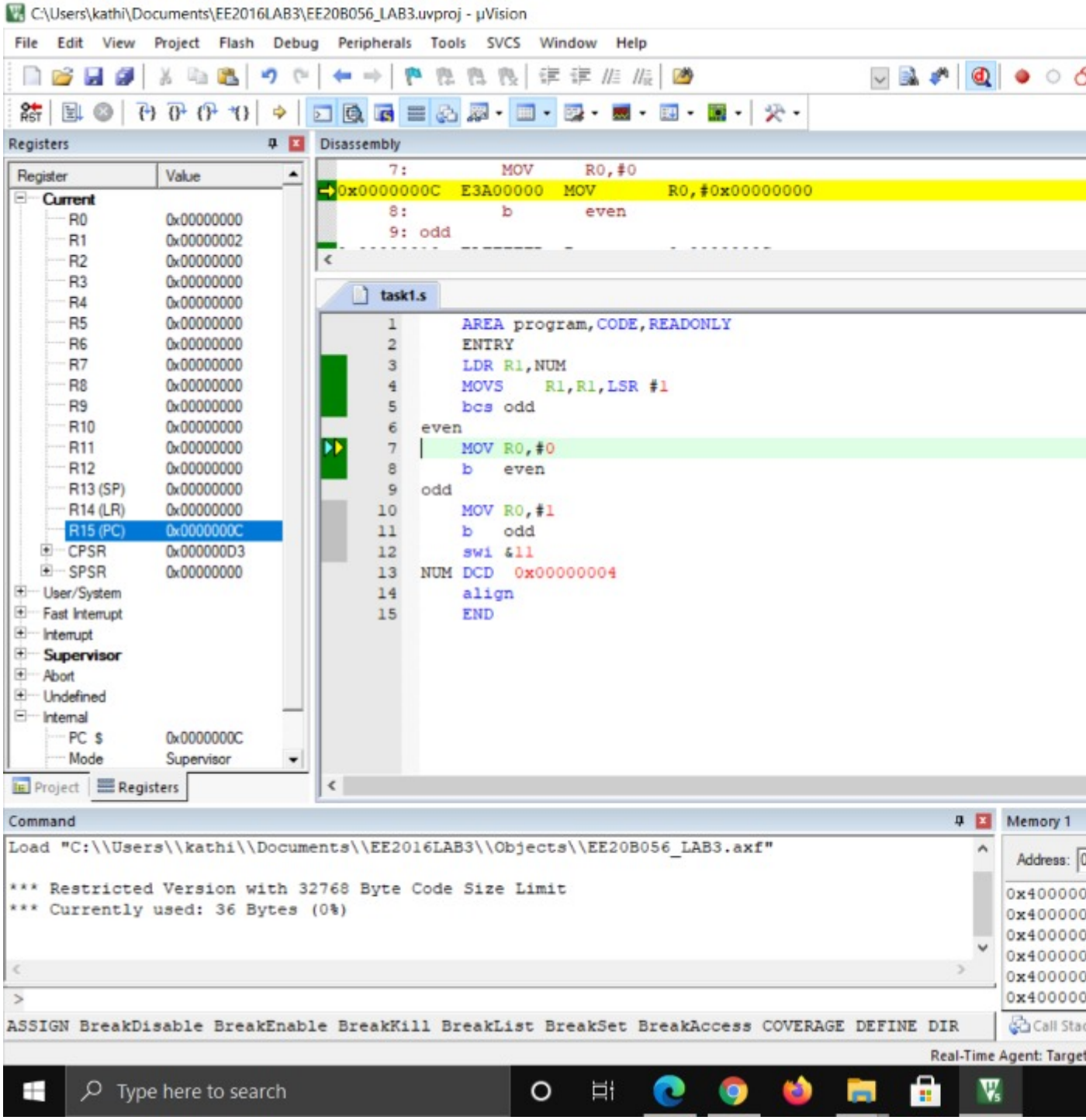
R2 – used to take the lower byte one by one by masking with R0 with R5

R5 – 0x0000000F – mask, when masked with this produces 4 bit LSBs alone

The above registers are used as mentioned in the comments in the code

TASK 3 – ODD OR EVEN:

Given a 32-bit number, identify whether it is an even or odd. (You implementation should not involve division).



CODE:

```
AREA program,CODE,READONLY

ENTRY

LDR R1,NUM
```

```
        MOVS   R1,R1,LSR #1
        bcs odd
even
        MOV    R0,#0
        b      even
odd
        MOV    R0,#1
        b      odd
        swi    &11
NUM DCD  0x00000004
        align
        END
```

LOGIC:

Given number is taken in R1

R1 is logically shifted to right , if it is odd carry is set, if it is even carry is not set – in both cases it jumps to either of the loops corresponding to the case.