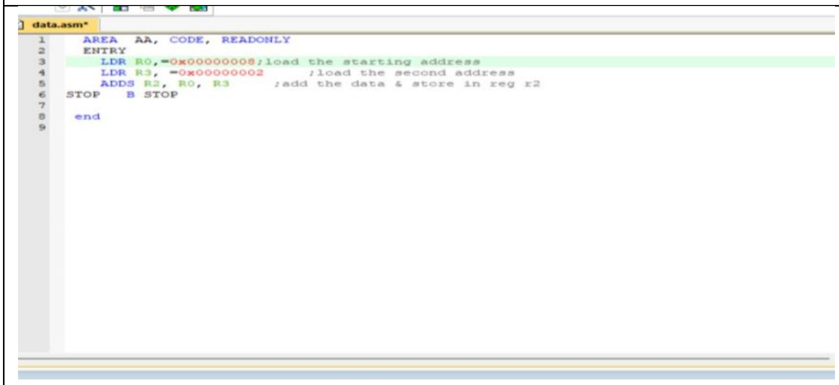
KLE Technological University
School of Electronics and Communication Engineering

ARM Microcontroller Lab (23EEEP202)

Team -6
Srikrishna 459
Niharika 435
Karthik 413
Mahanthesh 406

Expt./ Job No.1: Write an ALP to achieve the following arithmetic operations:

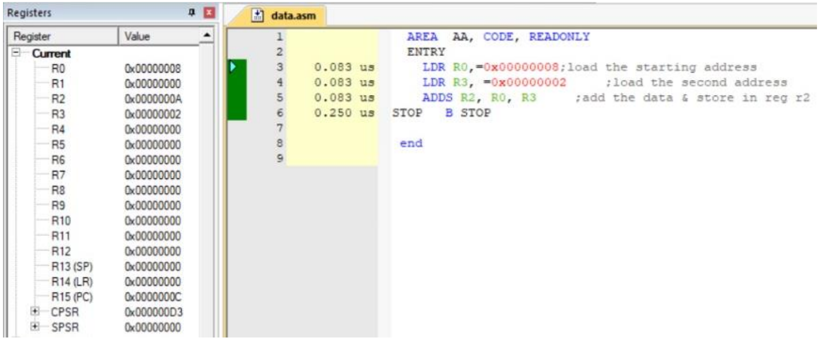
i. Subtraction



data.asm*

```
   AREA  AA, CODE, READONLY
   ENTRY
     LDR R0,=0x00000008;load the starting address
     LDR R3, =0x00000002       ;load the second address
     ADDS R2, R0, R3        ;add the data & store in reg r2
STOP    B STOP

 end
```

Output:



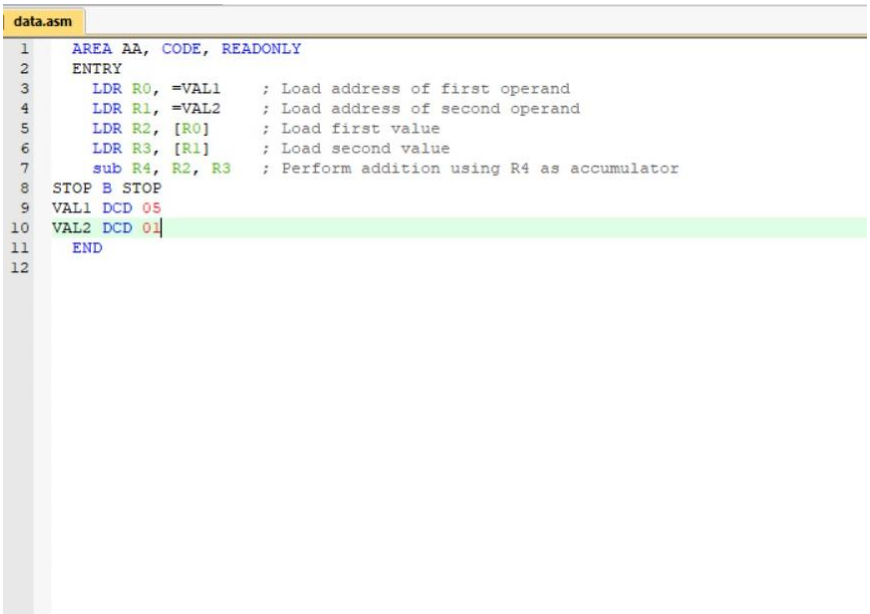
Registers

Register	Value
Current	
R0	0x00000008
R1	0x00000000
R2	0x0000000A
R3	0x00000002
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x0000000C
CPSR	0x000000D3
SPSR	0x00000000

data.asm

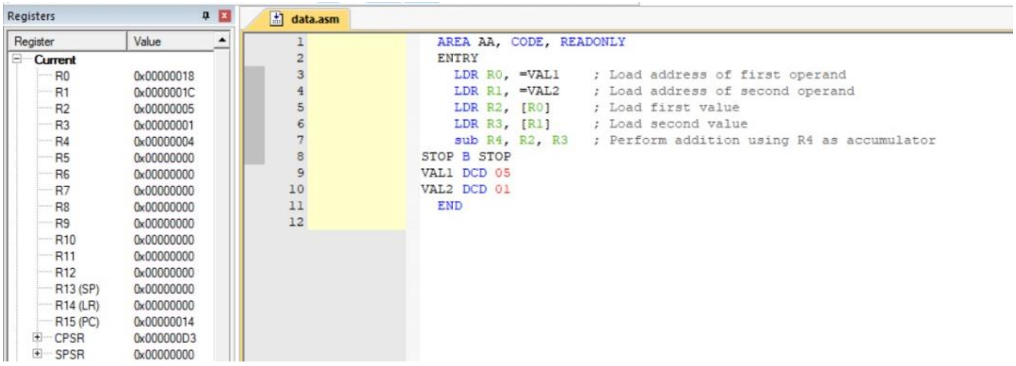
```
                AREA  AA, CODE, READONLY
                ENTRY
   0.083 us       LDR R0,=0x00000008;load the starting address
   0.083 us       LDR R3, =0x00000002      ;load the second address
   0.083 us       ADDS R2, R0, R3      ;add the data & store in reg r2
   0.250 us     STOP    B STOP

               end
```

Alternate code:



```
  AREA AA, CODE, READONLY
  ENTRY
    LDR R0, =VAL1     ; Load address of first operand
    LDR R1, =VAL2     ; Load address of second operand
    LDR R2, [R0]      ; Load first value
    LDR R3, [R1]      ; Load second value
    sub R4, R2, R3    ; Perform addition using R4 as accumulator
STOP B STOP
VAL1 DCD 05
VAL2 DCD 01
  END
```

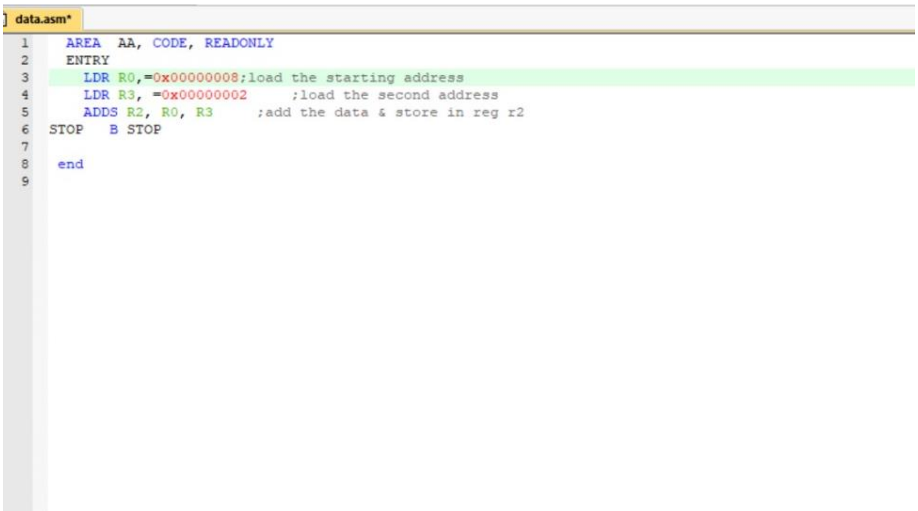
Output:



Register	Value
Current	
R0	0x00000018
R1	0x0000001C
R2	0x00000005
R3	0x00000001
R4	0x00000004
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000014
CPSR	0x000000D3
SPSR	0x00000000

```
  AREA AA, CODE, READONLY
  ENTRY
    LDR R0, =VAL1     ; Load address of first operand
    LDR R1, =VAL2     ; Load address of second operand
    LDR R2, [R0]      ; Load first value
    LDR R3, [R1]      ; Load second value
    sub R4, R2, R3    ; Perform addition using R4 as accumulator
STOP B STOP
VAL1 DCD 05
VAL2 DCD 01
  END
```

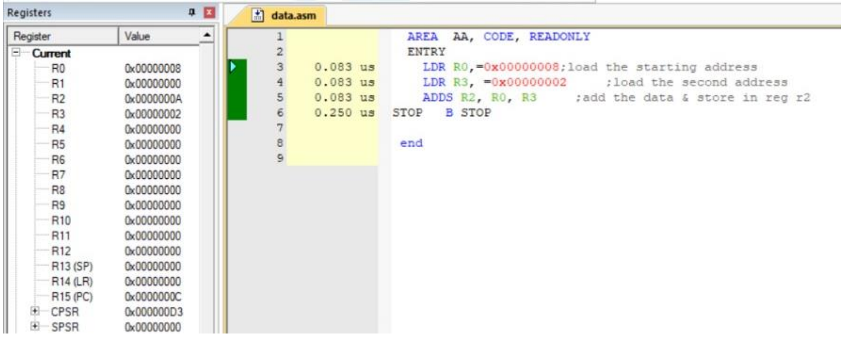
ii. 32 bit addition



data.asm*

```
   AREA  AA, CODE, READONLY
   ENTRY
     LDR R0,=0x00000008;load the starting address
     LDR R3, =0x00000002      ;load the second address
     ADDS R2, R0, R3      ;add the data & store in reg r2
STOP    B STOP

 end
```

Output:



Registers

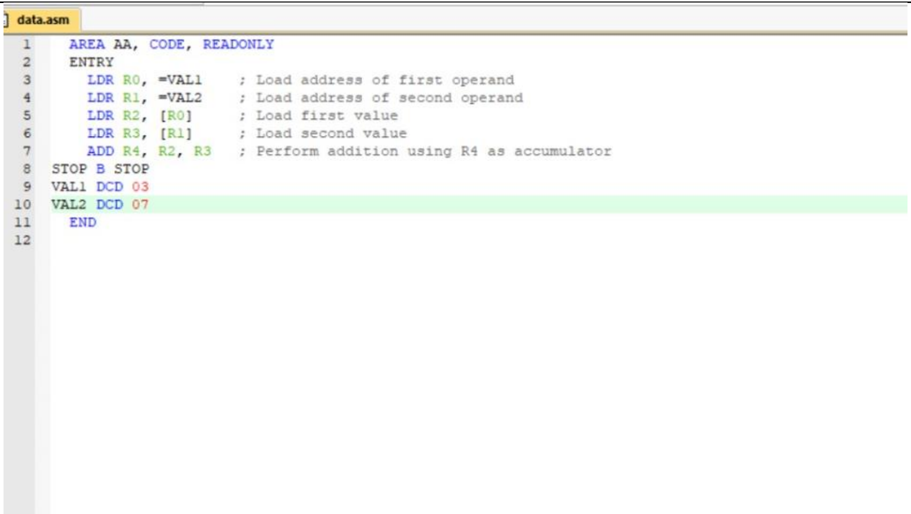
```
Register      Value
Current
  R0          0x00000008
  R1          0x00000000
  R2          0x0000000A
  R3          0x00000002
  R4          0x00000000
  R5          0x00000000
  R6          0x00000000
  R7          0x00000000
  R8          0x00000000
  R9          0x00000000
  R10         0x00000000
  R11         0x00000000
  R12         0x00000000
  R13 (SP)    0x00000000
  R14 (LR)    0x00000000
  R15 (PC)    0x0000000C
  CPSR        0x000000D3
  SPSR        0x00000000
```

data.asm

```
             AREA  AA, CODE, READONLY
             ENTRY
0.083 us       LDR R0,=0x00000008;load the starting address
0.083 us       LDR R3, =0x00000002      ;load the second address
0.083 us       ADDS R2, R0, R3      ;add the data & store in reg r2
0.250 us  STOP    B STOP

           end
```

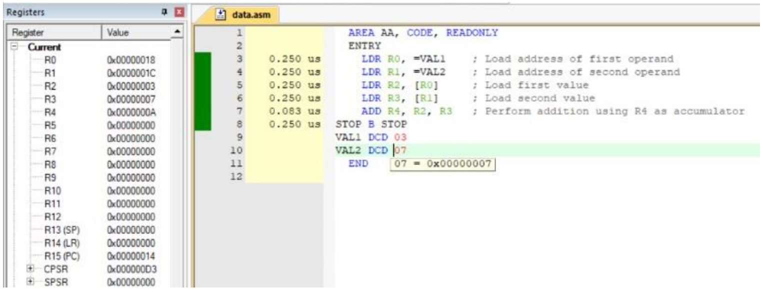
Alternate code:

ARM Microcontroller Lab (23EEEP202)



```
    AREA AA, CODE, READONLY
    ENTRY
      LDR R0, =VAL1     ; Load address of first operand
      LDR R1, =VAL2     ; Load address of second operand
      LDR R2, [R0]      ; Load first value
      LDR R3, [R1]      ; Load second value
      ADD R4, R2, R3    ; Perform addition using R4 as accumulator
  STOP B STOP
  VAL1 DCD 03
  VAL2 DCD 07
    END
```

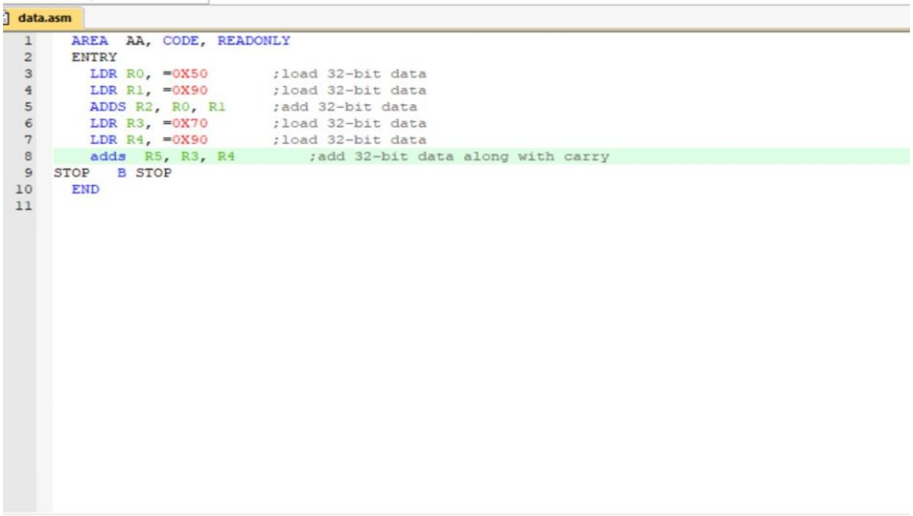
Output:



Register	Value
Current	
R0	0x00000018
R1	0x0000001C
R2	0x00000003
R3	0x00000007
R4	0x0000000A
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000014
CPSR	0x000000D3
SPSR	0x00000000

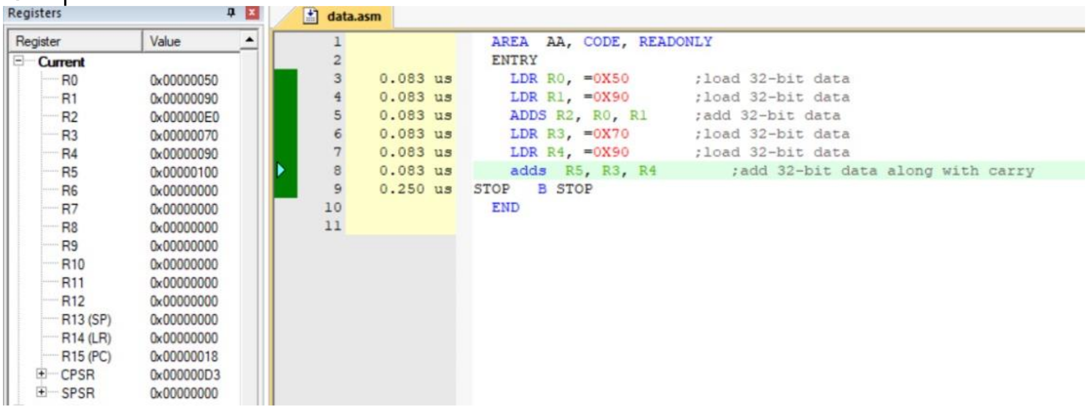
```
                 AREA AA, CODE, READONLY
                 ENTRY
    0.250 us       LDR R0, =VAL1     ; Load address of first operand
    0.250 us       LDR R1, =VAL2     ; Load address of second operand
    0.250 us       LDR R2, [R0]      ; Load first value
    0.250 us       LDR R3, [R1]      ; Load second value
    0.083 us       ADD R4, R2, R3    ; Perform addition using R4 as accumulator
    0.250 us   STOP B STOP
               VAL1 DCD 03
               VAL2 DCD 07
                 END     07 = 0x00000007
```

iii. 64 bit addition



```
    AREA  AA, CODE, READONLY
    ENTRY
      LDR R0, =0X50         ;load 32-bit data
      LDR R1, =0X90         ;load 32-bit data
      ADDS R2, R0, R1       ;add 32-bit data
      LDR R3, =0X70         ;load 32-bit data
      LDR R4, =0X90         ;load 32-bit data
      adds  R5, R3, R4          ;add 32-bit data along with carry
  STOP    B STOP
    END
```

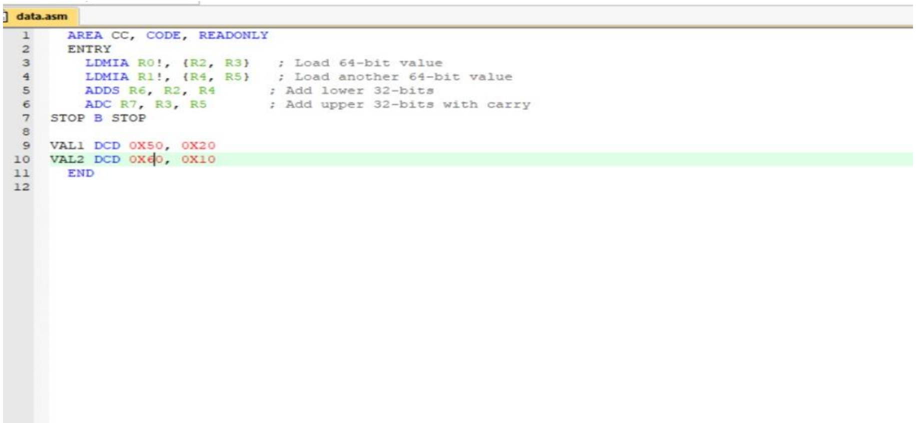
Output:



Register	Value
Current	
R0	0x00000050
R1	0x00000090
R2	0x000000E0
R3	0x00000070
R4	0x00000090
R5	0x00000100
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000018
CPSR	0x000000D3
SPSR	0x00000000

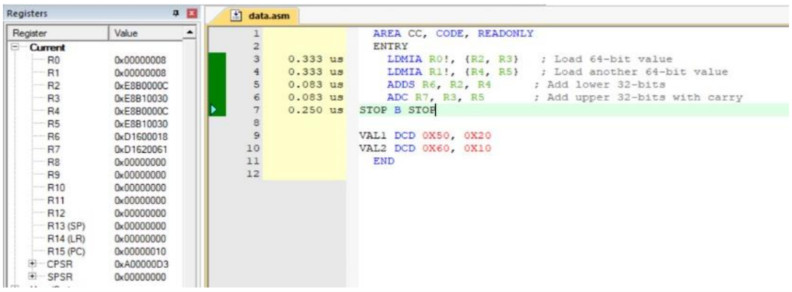
```
                  AREA  AA, CODE, READONLY
                  ENTRY
    0.083 us        LDR R0, =0X50         ;load 32-bit data
    0.083 us        LDR R1, =0X90         ;load 32-bit data
    0.083 us        ADDS R2, R0, R1       ;add 32-bit data
    0.083 us        LDR R3, =0X70         ;load 32-bit data
    0.083 us        LDR R4, =0X90         ;load 32-bit data
    0.083 us        adds  R5, R3, R4          ;add 32-bit data along with carry
    0.250 us      STOP    B STOP
                  END
```

Alternate code:



```
    AREA CC, CODE, READONLY
    ENTRY
      LDMIA R0!, {R2, R3}    ; Load 64-bit value
      LDMIA R1!, {R4, R5}    ; Load another 64-bit value
      ADDS R6, R2, R4       ; Add lower 32-bits
      ADC R7, R3, R5        ; Add upper 32-bits with carry
  STOP B STOP

  VAL1 DCD 0X50, 0X20
  VAL2 DCD 0X60, 0X10
    END
```

Output:



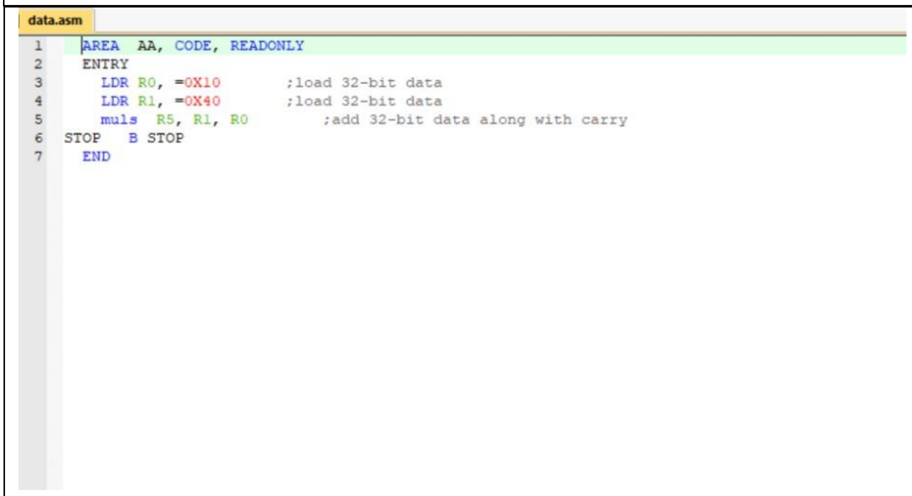
Registers

Register	Value
Current	
R0	0x00000008
R1	0x00000008
R2	0xE8B0000C
R3	0xE8B10030
R4	0xE8B0000C
R5	0xE8B10030
R6	0xD1600018
R7	0xD1620061
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000010
CPSR	0xA00000D3
SPSR	0x00000000

data.asm

```
             AREA CC, CODE, READONLY
             ENTRY
0.333 us       LDMIA R0!, {R2, R3}    ; Load 64-bit value
0.333 us       LDMIA R1!, {R4, R5}    ; Load another 64-bit value
0.083 us       ADDS R6, R2, R4       ; Add lower 32-bits
0.083 us       ADC R7, R3, R5        ; Add upper 32-bits with carry
0.250 us   STOP B STOP

           VAL1 DCD 0X50, 0X20
           VAL2 DCD 0X60, 0X10
             END
```

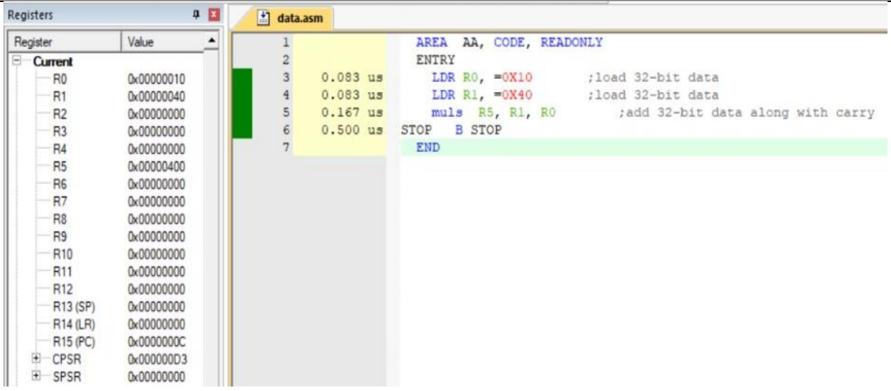
iv. Multiplication



data.asm

```
    AREA  AA, CODE, READONLY
    ENTRY
      LDR R0, =0X10       ;load 32-bit data
      LDR R1, =0X40       ;load 32-bit data
      muls  R5, R1, R0        ;add 32-bit data along with carry
STOP    B STOP
    END
```

Output:



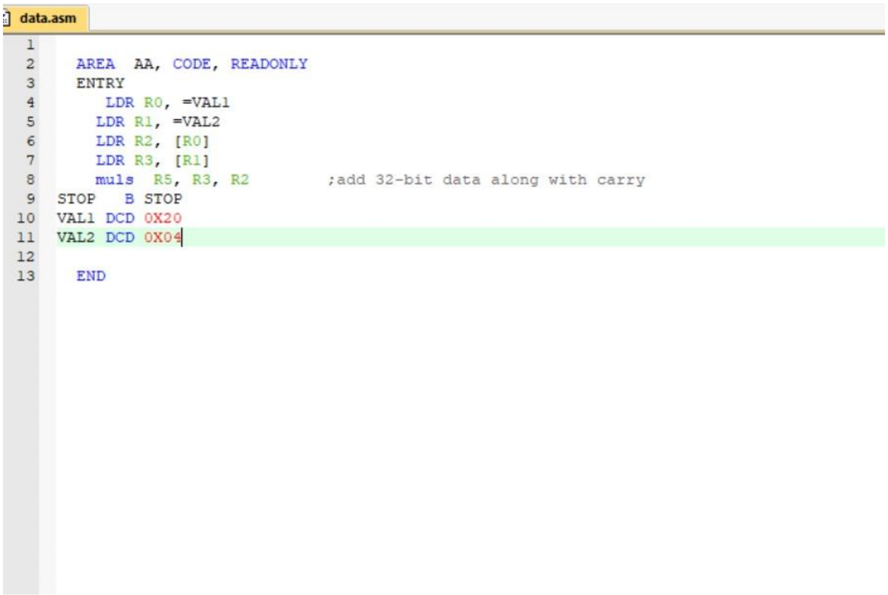
Registers

Register	Value
Current	
R0	0x00000010
R1	0x00000040
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000400
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x0000000C
CPSR	0x000000D3
SPSR	0x00000000

data.asm

```
               AREA  AA, CODE, READONLY
               ENTRY
  0.083 us       LDR R0, =0X10       ;load 32-bit data
  0.083 us       LDR R1, =0X40       ;load 32-bit data
  0.167 us       muls  R5, R1, R0        ;add 32-bit data along with carry
  0.500 us   STOP    B STOP
               END
```

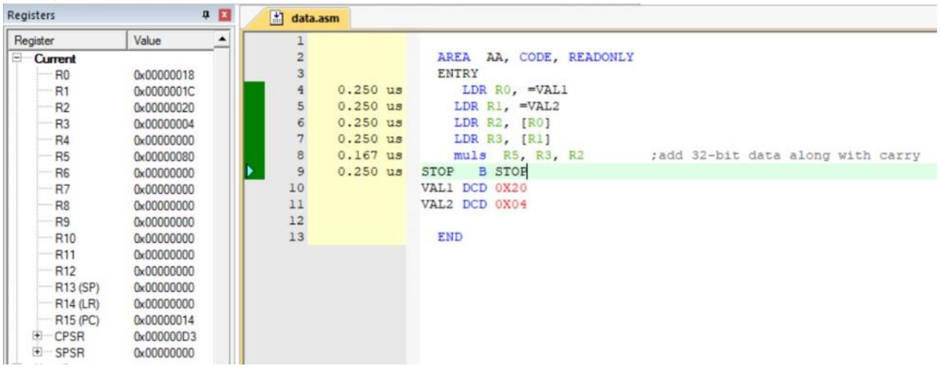
Alternate code:



data.asm

```
  AREA  AA, CODE, READONLY
  ENTRY
     LDR R0, =VAL1
    LDR R1, =VAL2
    LDR R2, [R0]
    LDR R3, [R1]
    muls  R5, R3, R2         ;add 32-bit data along with carry
STOP    B STOP
VAL1 DCD 0X20
VAL2 DCD 0X04

  END
```

Output:



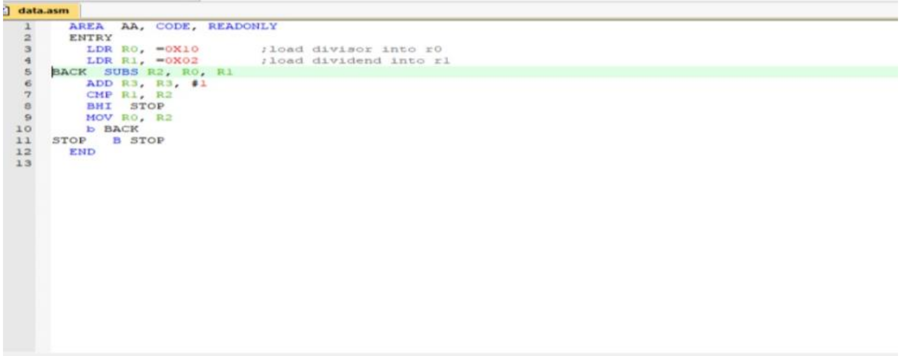
Registers

Register	Value
Current	
R0	0x00000018
R1	0x0000001C
R2	0x00000020
R3	0x00000004
R4	0x00000000
R5	0x00000080
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000014
CPSR	0x000000D3
SPSR	0x00000000

data.asm

```
                AREA  AA, CODE, READONLY
                ENTRY
0.250 us           LDR R0, =VAL1
0.250 us          LDR R1, =VAL2
0.250 us          LDR R2, [R0]
0.250 us          LDR R3, [R1]
0.167 us          muls  R5, R3, R2         ;add 32-bit data along with carry
0.250 us      STOP    B STOP
              VAL1 DCD 0X20
              VAL2 DCD 0X04

                END
```

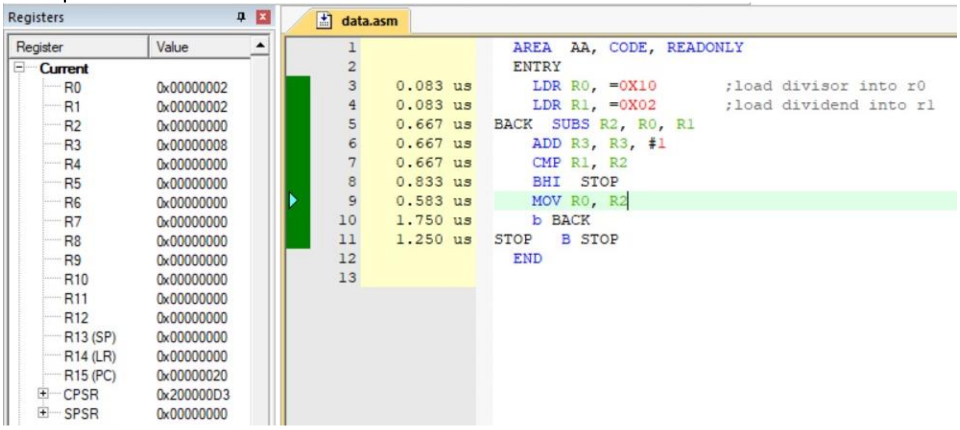
v. 32 bit binary divide



data.asm

```
    AREA   AA, CODE, READONLY
    ENTRY
       LDR R0, =0X10          ;load divisor into r0
       LDR R1, =0X02          ;load dividend into r1
BACK   SUBS R2, R0, R1
       ADD R3, R3, #1
       CMP R1, R2
       BHI   STOP
       MOV R0, R2
       b BACK
STOP    B STOP
    END
```

Output:

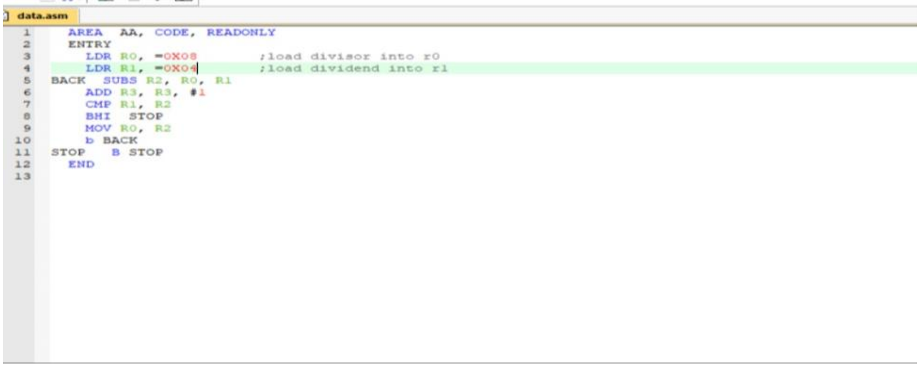


Register	Value
Current	
R0	0x00000002
R1	0x00000002
R2	0x00000000
R3	0x00000008
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000020
CPSR	0x200000D3
SPSR	0x00000000

data.asm

```
                    AREA   AA, CODE, READONLY
                    ENTRY
0.083 us               LDR R0, =0X10          ;load divisor into r0
0.083 us               LDR R1, =0X02          ;load dividend into r1
0.667 us        BACK   SUBS R2, R0, R1
0.667 us               ADD R3, R3, #1
0.667 us               CMP R1, R2
0.833 us               BHI   STOP
0.583 us               MOV R0, R2
1.750 us               b BACK
1.250 us        STOP    B STOP
                    END
```

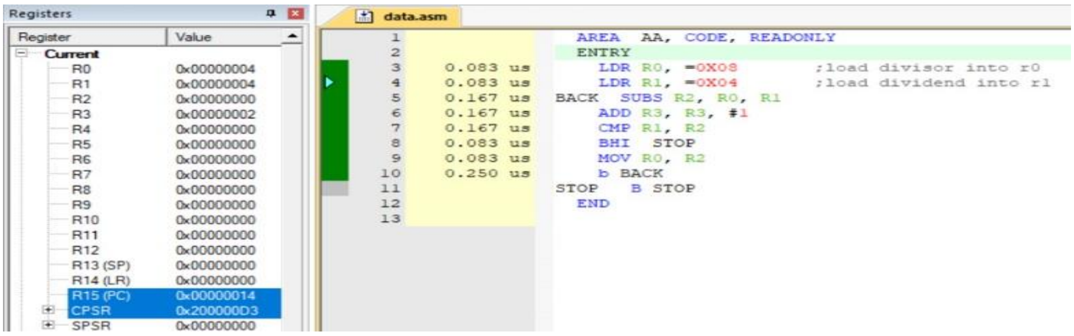
Alternate code:



data.asm

```
    AREA   AA, CODE, READONLY
    ENTRY
       LDR R0, =0X08            ;load divisor into r0
       LDR R1, =0X04            ;load dividend into r1
BACK   SUBS R2, R0, R1
       ADD R3, R3, #1
       CMP R1, R2
       BHI  STOP
       MOV R0, R2
       b BACK
STOP    B STOP
    END
```

Output:



Registers

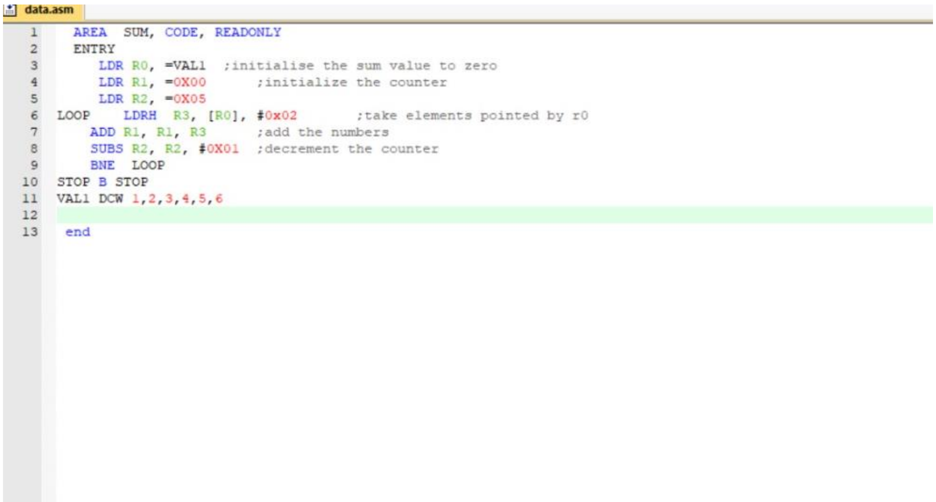
Register	Value
Current	
R0	0x00000004
R1	0x00000004
R2	0x00000000
R3	0x00000002
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000014
CPSR	0x200000D3
SPSR	0x00000000

data.asm

```
                 AREA   AA, CODE, READONLY
                 ENTRY
0.083 us            LDR R0, =0X08            ;load divisor into r0
0.083 us            LDR R1, =0X04            ;load dividend into r1
0.167 us    BACK   SUBS R2, R0, R1
0.167 us            ADD R3, R3, #1
0.167 us            CMP R1, R2
0.083 us            BHI  STOP
0.083 us            MOV R0, R2
0.250 us            b BACK
            STOP    B STOP
                 END
```

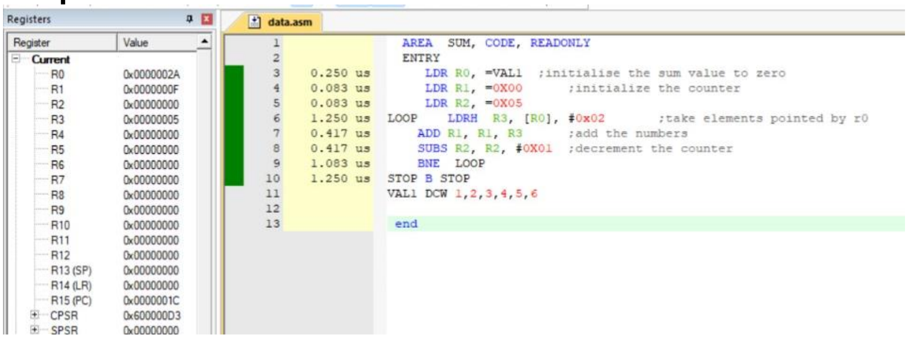
Expt./ Job No.2: **Write an ALP for the following using loops:**

i. Find the sum of ‘N’ 16 bit numbers



```
   AREA  SUM, CODE, READONLY
   ENTRY
      LDR R0, =VAL1  ;initialise the sum value to zero
      LDR R1, =0X00      ;initialize the counter
      LDR R2, =0X05
 LOOP    LDRH  R3, [R0], #0x02        ;take elements pointed by r0
     ADD R1, R1, R3      ;add the numbers
     SUBS R2, R2, #0X01  ;decrement the counter
     BNE  LOOP
 STOP B STOP
 VAL1 DCW 1,2,3,4,5,6

  end
```

Output:



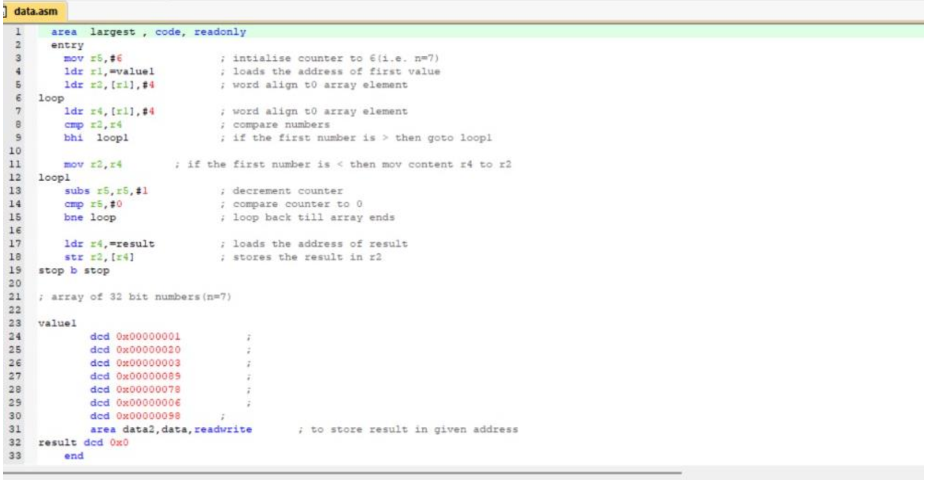
```
Register      Value
Current
  R0          0x0000002A
  R1          0x0000000F
  R2          0x00000000
  R3          0x00000005
  R4          0x00000000
  R5          0x00000000
  R6          0x00000000
  R7          0x00000000
  R8          0x00000000
  R9          0x00000000
  R10         0x00000000
  R11         0x00000000
  R12         0x00000000
  R13 (SP)    0x00000000
  R14 (LR)    0x00000000
  R15 (PC)    0x0000001C
  CPSR        0x600000D3
  SPSR        0x00000000
```

```
                 AREA  SUM, CODE, READONLY
                 ENTRY
   0.250 us        LDR R0, =VAL1  ;initialise the sum value to zero
   0.083 us        LDR R1, =0X00      ;initialize the counter
   0.083 us        LDR R2, =0X05
   1.250 us    LOOP    LDRH  R3, [R0], #0x02        ;take elements pointed by r0
   0.417 us       ADD R1, R1, R3      ;add the numbers
   0.417 us       SUBS R2, R2, #0X01  ;decrement the counter
   1.083 us       BNE  LOOP
   1.250 us    STOP B STOP
               VAL1 DCW 1,2,3,4,5,6

                 end
```

Alternate code:

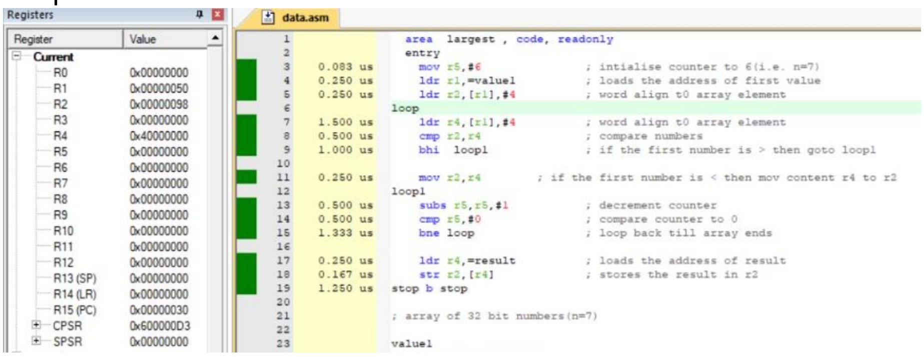
ii. Find the maximum/minimum of N numbers

A.Maximum:



```
  area  largest , code, readonly
  entry
    mov r5,#6                 ; intialise counter to 6(i.e. n=7)
    ldr r1,=value1            ; loads the address of first value
    ldr r2,[r1],#4            ; word align t0 array element
loop
    ldr r4,[r1],#4            ; word align t0 array element
    cmp r2,r4                 ; compare numbers
    bhi  loop1                ; if the first number is > then goto loop1

    mov r2,r4           ; if the first number is < then mov content r4 to r2
loop1
    subs r5,r5,#1             ; decrement counter
    cmp r5,#0                 ; compare counter to 0
    bne loop                  ; loop back till array ends

    ldr r4,=result            ; loads the address of result
    str r2,[r4]               ; stores the result in r2
stop b stop

; array of 32 bit numbers(n=7)

value1
        dcd 0x00000001          ;
        dcd 0x00000020          ;
        dcd 0x00000003          ;
        dcd 0x00000089          ;
        dcd 0x00000078          ;
        dcd 0x00000006          ;
        dcd 0x00000098      ;
        area data2,data,readwrite       ; to store result in given address
result dcd 0x0
    end
```

Output:

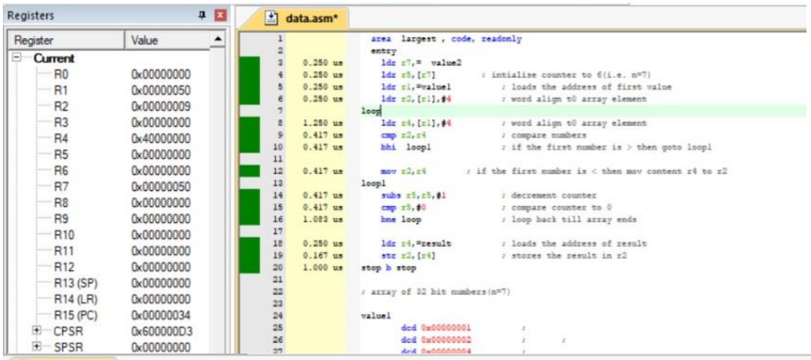


Register	Value
Current	
R0	0x00000000
R1	0x00000050
R2	0x00000098
R3	0x00000000
R4	0x40000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000030
CPSR	0x600000D3
SPSR	0x00000000

```
              area  largest , code, readonly
              entry
  0.083 us      mov r5,#6                 ; intialise counter to 6(i.e. n=7)
  0.250 us      ldr r1,=value1            ; loads the address of first value
  0.250 us      ldr r2,[r1],#4            ; word align t0 array element
            loop
  1.500 us      ldr r4,[r1],#4            ; word align t0 array element
  0.500 us      cmp r2,r4                 ; compare numbers
  1.000 us      bhi  loop1                ; if the first number is > then goto loop1

  0.250 us      mov r2,r4           ; if the first number is < then mov content r4 to r2
            loop1
  0.500 us      subs r5,r5,#1             ; decrement counter
  0.500 us      cmp r5,#0                 ; compare counter to 0
  1.333 us      bne loop                  ; loop back till array ends

  0.250 us      ldr r4,=result            ; loads the address of result
  0.167 us      str r2,[r4]               ; stores the result in r2
  1.250 us  stop b stop

            ; array of 32 bit numbers(n=7)

            value1
```

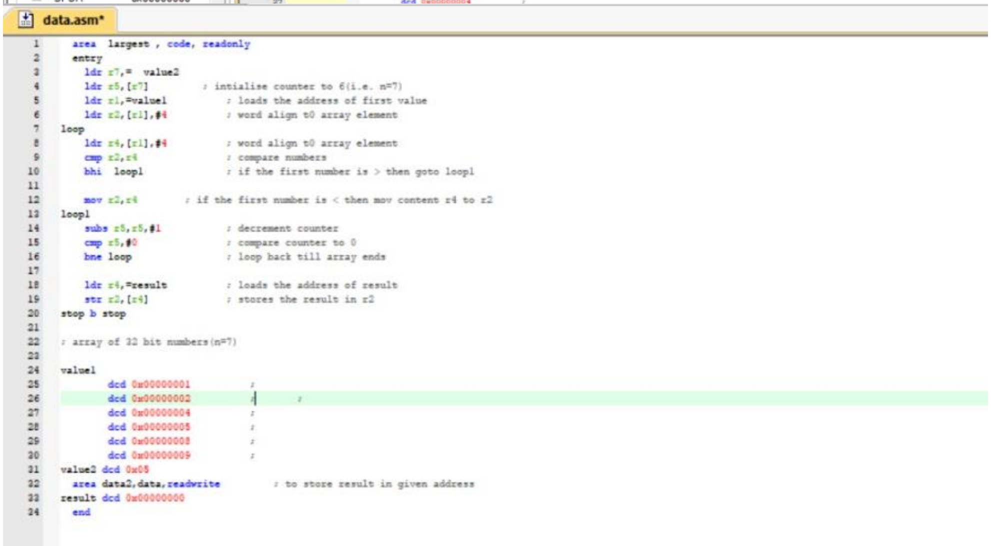
ARM Microcontroller Lab (23EEEP202)

Alternate code:



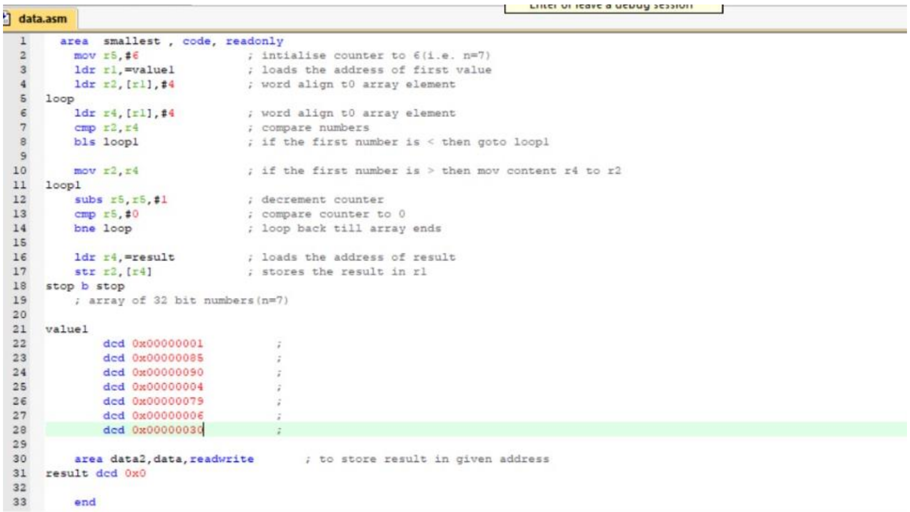
Register	Value
Current	
R0	0x00000000
R1	0x00000050
R2	0x00000009
R3	0x00000000
R4	0x40000000
R5	0x00000000
R6	0x00000000
R7	0x00000050
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000034
CPSR	0x600000D3
SPSR	0x00000000

```
              area  largest , code, readonly
              entry
0.250 us        ldr r7,=  value2
0.250 us        ldr r5,[r7]         ; intialise counter to 6(i.e. n=7)
0.250 us        ldr r1,=value1          ; loads the address of first value
0.250 us        ldr r2,[r1],#4          ; word align t0 array element
            loop
1.250 us        ldr r4,[r1],#4          ; word align t0 array element
0.417 us        cmp r2,r4               ; compare numbers
0.417 us        bhi  loop1              ; if the first number is > then goto loop1

0.417 us        mov r2,r4         ; if the first number is < then mov content r4 to r2
            loop1
0.417 us        subs r5,r5,#1           ; decrement counter
0.417 us        cmp r5,#0               ; compare counter to 0
1.083 us        bne loop                ; loop back till array ends

0.250 us        ldr r4,=result          ; loads the address of result
0.167 us        str r2,[r4]             ; stores the result in r2
1.000 us    stop b stop

            ; array of 32 bit numbers(n=7)

            value1
                    dcd 0x00000001          ;
                    dcd 0x00000002          ;       ;
```



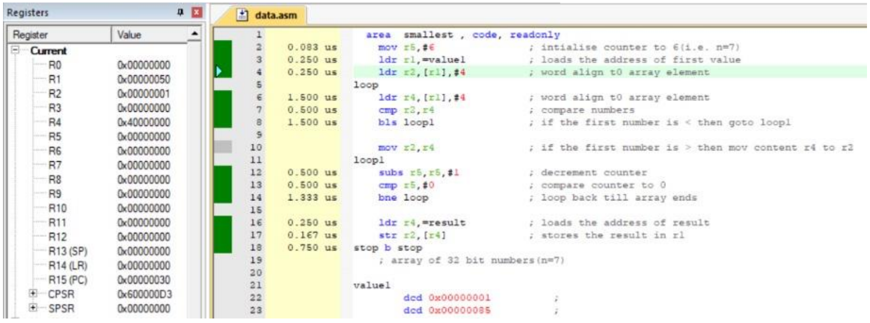
```
  area  largest , code, readonly
  entry
    ldr r7,=  value2
    ldr r5,[r7]         ; intialise counter to 6(i.e. n=7)
    ldr r1,=value1          ; loads the address of first value
    ldr r2,[r1],#4          ; word align t0 array element
loop
    ldr r4,[r1],#4          ; word align t0 array element
    cmp r2,r4               ; compare numbers
    bhi  loop1              ; if the first number is > then goto loop1

    mov r2,r4         ; if the first number is < then mov content r4 to r2
loop1
    subs r5,r5,#1           ; decrement counter
    cmp r5,#0               ; compare counter to 0
    bne loop                ; loop back till array ends

    ldr r4,=result          ; loads the address of result
    str r2,[r4]             ; stores the result in r2
stop b stop

; array of 32 bit numbers(n=7)

value1
        dcd 0x00000001          ;
        dcd 0x00000002          ;       ;
        dcd 0x00000004          ;
        dcd 0x00000005          ;
        dcd 0x00000008          ;
        dcd 0x00000009          ;
value2 dcd 0x05
  area data2,data,readwrite           ; to store result in given address
result dcd 0x00000000
  end
```

B. MINIMUM:



```
  area  smallest , code, readonly
    mov r5,#6                   ; intialise counter to 6(i.e. n=7)
    ldr r1,=value1              ; loads the address of first value
    ldr r2,[r1],#4              ; word align t0 array element
loop
    ldr r4,[r1],#4              ; word align t0 array element
    cmp r2,r4                   ; compare numbers
    bls loop1                   ; if the first number is < then goto loop1

    mov r2,r4                   ; if the first number is > then mov content r4 to r2
loop1
    subs r5,r5,#1               ; decrement counter
    cmp r5,#0                   ; compare counter to 0
    bne loop                    ; loop back till array ends

    ldr r4,=result              ; loads the address of result
    str r2,[r4]                 ; stores the result in r1
stop b stop
    ; array of 32 bit numbers(n=7)

value1
        dcd 0x00000001          ;
        dcd 0x00000085          ;
        dcd 0x00000090          ;
        dcd 0x00000004          ;
        dcd 0x00000079          ;
        dcd 0x00000006          ;
        dcd 0x00000030          ;

    area data2,data,readwrite       ; to store result in given address
result dcd 0x0

    end
```

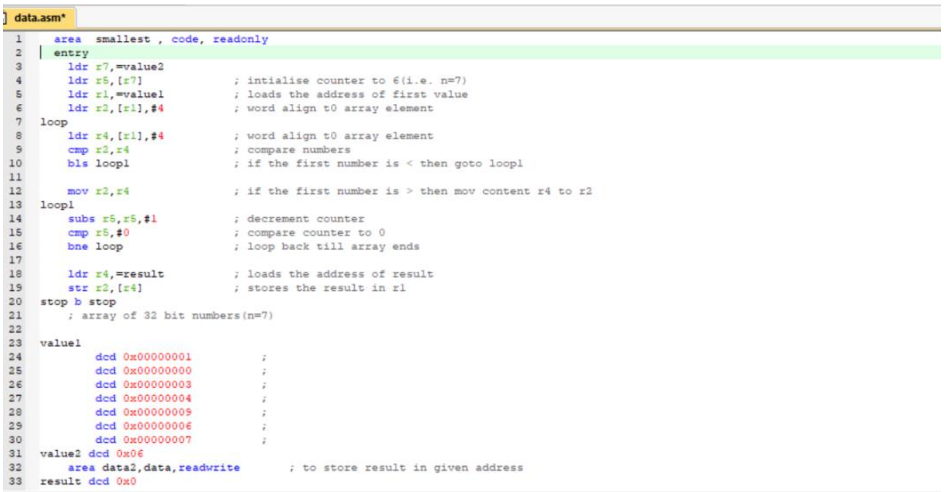
Output:



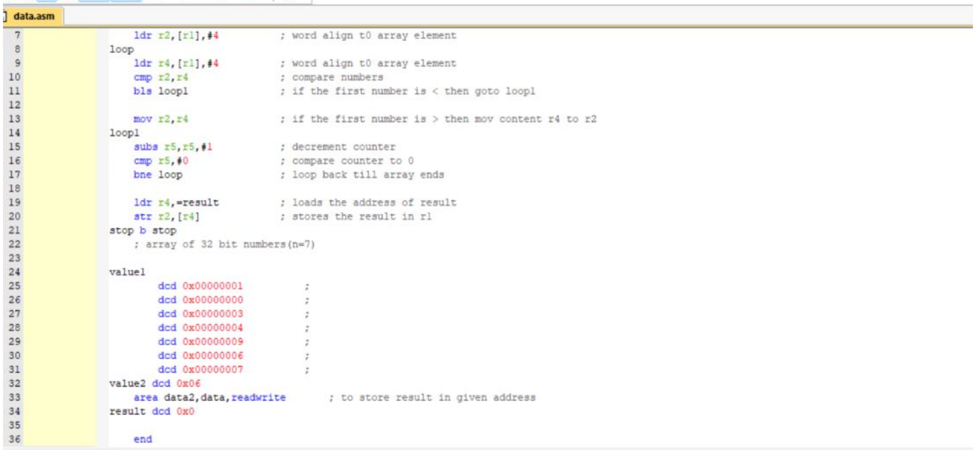
Register	Value
Current	
R0	0x00000000
R1	0x00000050
R2	0x00000001
R3	0x00000000
R4	0x40000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000030
CPSR	0x600000D3
SPSR	0x00000000

```
                  area  smallest , code, readonly
     0.083 us       mov r5,#6                ; intialise counter to 6(i.e. n=7)
     0.250 us       ldr r1,=value1           ; loads the address of first value
     0.250 us       ldr r2,[r1],#4           ; word align t0 array element
                 loop
     1.500 us       ldr r4,[r1],#4           ; word align t0 array element
     0.500 us       cmp r2,r4                ; compare numbers
     1.500 us       bls loop1                ; if the first number is < then goto loop1

                    mov r2,r4                ; if the first number is > then mov content r4 to r2
                 loop1
     0.500 us       subs r5,r5,#1            ; decrement counter
     0.500 us       cmp r5,#0                ; compare counter to 0
     1.333 us       bne loop                 ; loop back till array ends

     0.250 us       ldr r4,=result           ; loads the address of result
     0.167 us       str r2,[r4]              ; stores the result in r1
     0.750 us    stop b stop
                    ; array of 32 bit numbers(n=7)

                 value1
                        dcd 0x00000001       ;
                        dcd 0x00000085       ;
```

Alternate code:



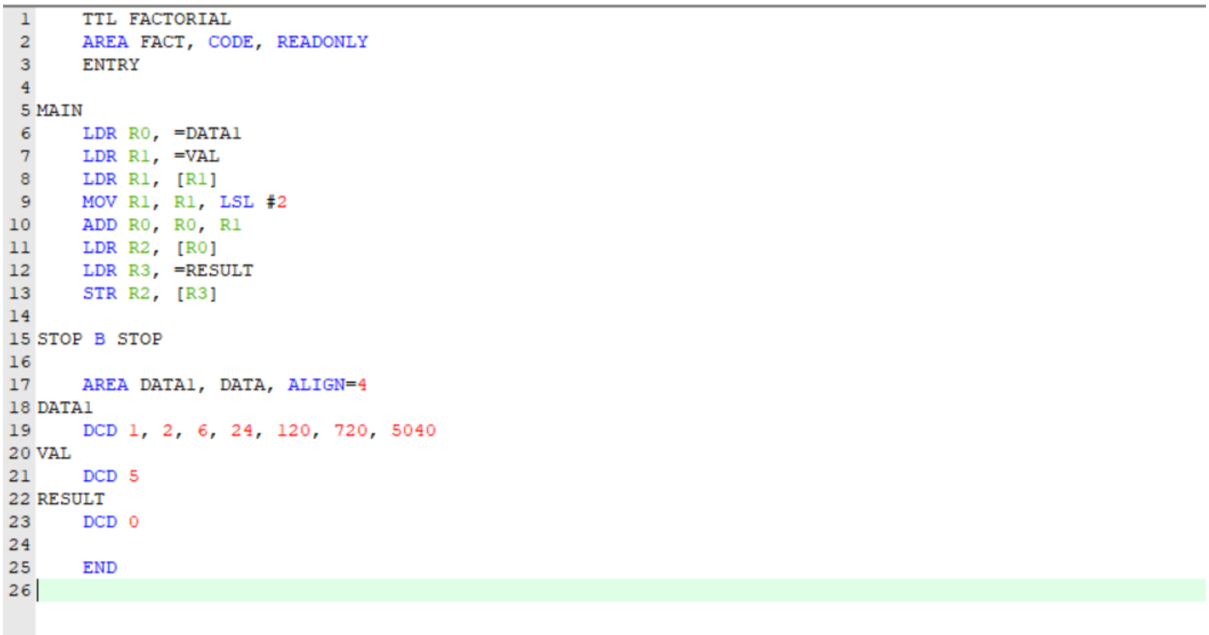
```
    area  smallest , code, readonly
    entry
      ldr r7,=value2
      ldr r5,[r7]             ; intialise counter to 6(i.e. n=7)
      ldr r1,=value1          ; loads the address of first value
      ldr r2,[r1],#4          ; word align t0 array element
  loop
      ldr r4,[r1],#4          ; word align t0 array element
      cmp r2,r4               ; compare numbers
      bls loop1               ; if the first number is < then goto loop1

      mov r2,r4               ; if the first number is > then mov content r4 to r2
  loop1
      subs r5,r5,#1           ; decrement counter
      cmp r5,#0               ; compare counter to 0
      bne loop                ; loop back till array ends

      ldr r4,=result          ; loads the address of result
      str r2,[r4]             ; stores the result in r1
  stop b stop
      ; array of 32 bit numbers(n=7)

  value1
          dcd 0x00000001          ;
          dcd 0x00000000          ;
          dcd 0x00000003          ;
          dcd 0x00000004          ;
          dcd 0x00000009          ;
          dcd 0x00000006          ;
          dcd 0x00000007          ;
  value2 dcd 0x06
      area data2,data,readwrite        ; to store result in given address
  result dcd 0x0
```



```
      ldr r2,[r1],#4          ; word align t0 array element
  loop
      ldr r4,[r1],#4          ; word align t0 array element
      cmp r2,r4               ; compare numbers
      bls loop1               ; if the first number is < then goto loop1

      mov r2,r4               ; if the first number is > then mov content r4 to r2
  loop1
      subs r5,r5,#1           ; decrement counter
      cmp r5,#0               ; compare counter to 0
      bne loop                ; loop back till array ends

      ldr r4,=result          ; loads the address of result
      str r2,[r4]             ; stores the result in r1
  stop b stop
      ; array of 32 bit numbers(n=7)

  value1
          dcd 0x00000001          ;
          dcd 0x00000000          ;
          dcd 0x00000003          ;
          dcd 0x00000004          ;
          dcd 0x00000009          ;
          dcd 0x00000006          ;
          dcd 0x00000007          ;
  value2 dcd 0x06
      area data2,data,readwrite        ; to store result in given address
  result dcd 0x0

      end
```

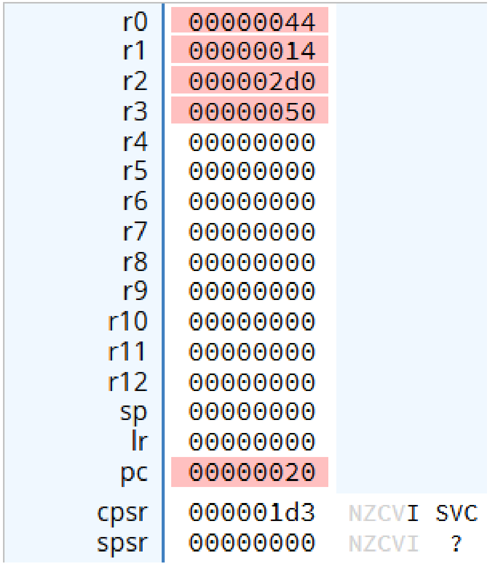
ii. Find the factorial of a given number with and without a look up table.
Apply suitable machine dependent optimization technique and analyze for memory and time consumed.

A. Write an ALP to find factorial of a given number with lookup table:



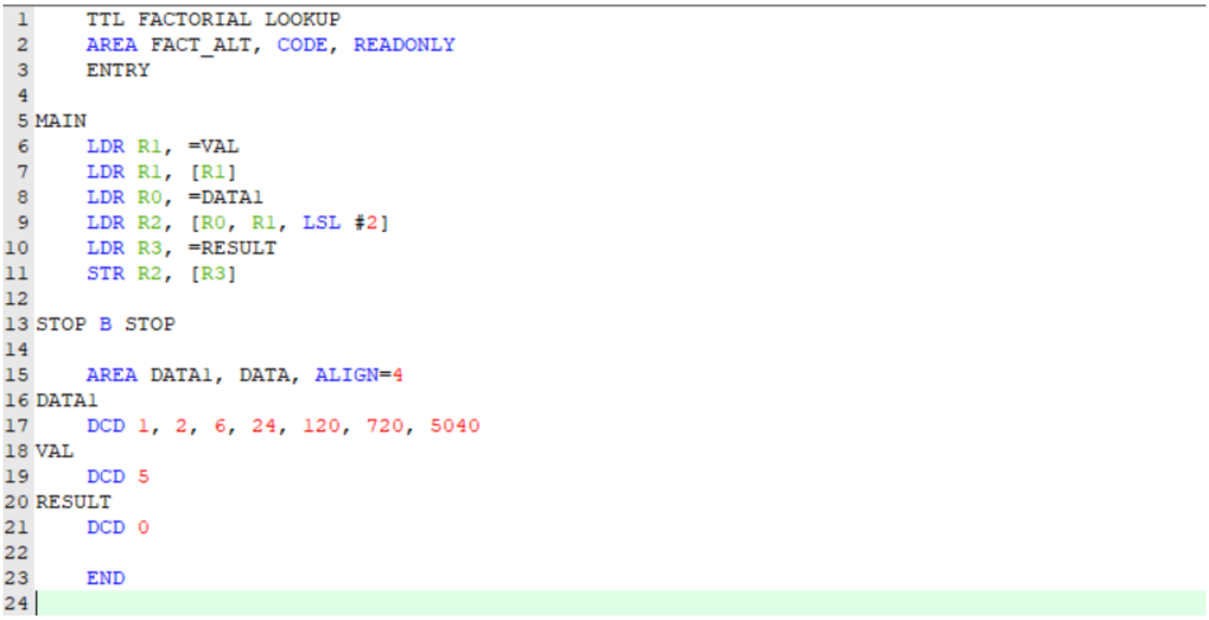
```
    TTL FACTORIAL
    AREA FACT, CODE, READONLY
    ENTRY

MAIN
    LDR R0, =DATA1
    LDR R1, =VAL
    LDR R1, [R1]
    MOV R1, R1, LSL #2
    ADD R0, R0, R1
    LDR R2, [R0]
    LDR R3, =RESULT
    STR R2, [R3]

STOP B STOP

    AREA DATA1, DATA, ALIGN=4
DATA1
    DCD 1, 2, 6, 24, 120, 720, 5040
VAL
    DCD 5
RESULT
    DCD 0

    END

```

Output:



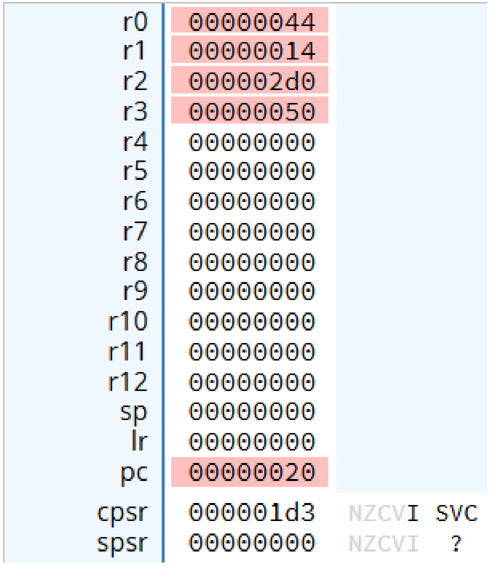
```
   r0   00000044
   r1   00000014
   r2   000002d0
   r3   00000050
   r4   00000000
   r5   00000000
   r6   00000000
   r7   00000000
   r8   00000000
   r9   00000000
  r10   00000000
  r11   00000000
  r12   00000000
   sp   00000000
   lr   00000000
   pc   00000020

 cpsr   000001d3   NZCVI SVC
 spsr   00000000   NZCVI  ?
```

Alternate code:



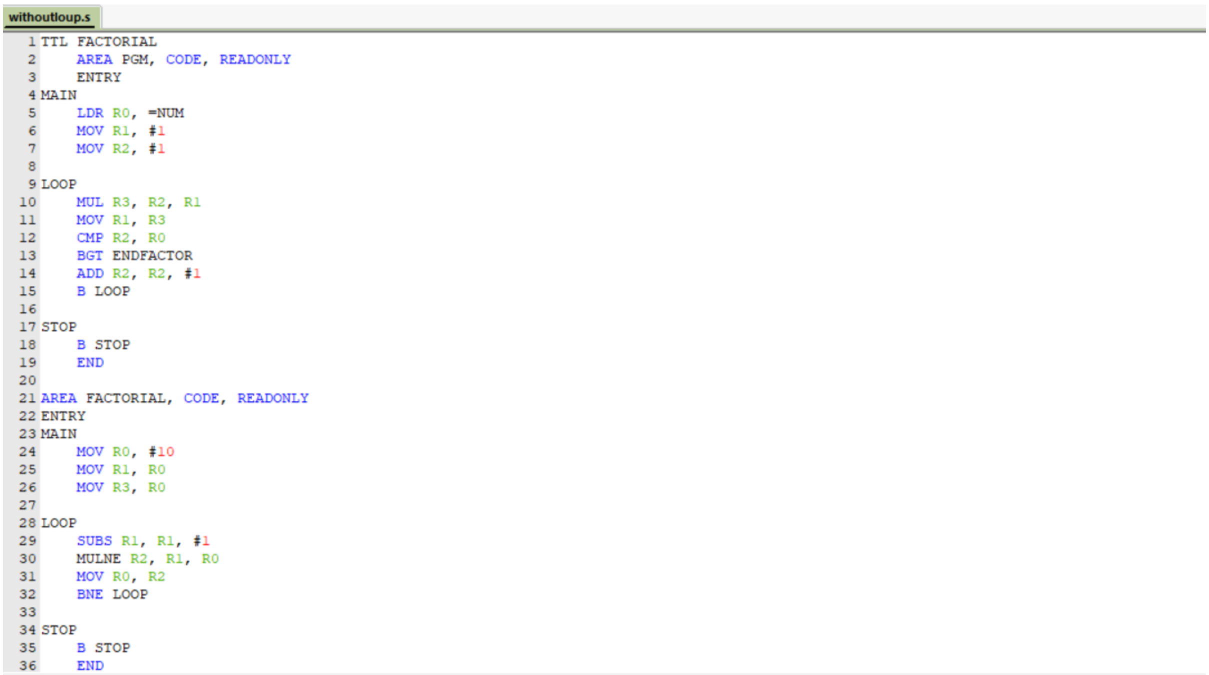
```
      TTL FACTORIAL LOOKUP
      AREA FACT_ALT, CODE, READONLY
      ENTRY

MAIN
      LDR R1, =VAL
      LDR R1, [R1]
      LDR R0, =DATA1
      LDR R2, [R0, R1, LSL #2]
      LDR R3, =RESULT
      STR R2, [R3]

STOP B STOP

      AREA DATA1, DATA, ALIGN=4
DATA1
      DCD 1, 2, 6, 24, 120, 720, 5040
VAL
      DCD 5
RESULT
      DCD 0

      END
```

Output:



```
   r0   00000044
   r1   00000014
   r2   000002d0
   r3   00000050
   r4   00000000
   r5   00000000
   r6   00000000
   r7   00000000
   r8   00000000
   r9   00000000
  r10   00000000
  r11   00000000
  r12   00000000
   sp   00000000
   lr   00000000
   pc   00000020
 cpsr   000001d3   NZCVI SVC
 spsr   00000000   NZCVI  ?
```

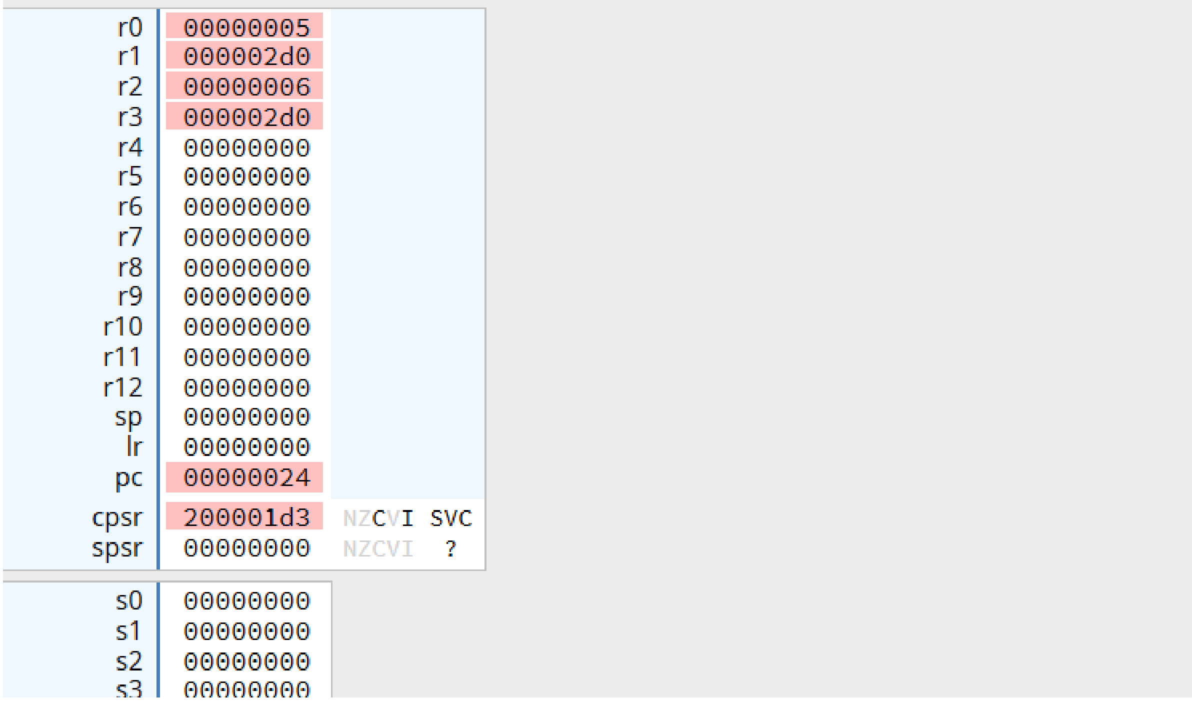
B. Write an ALP to find factorial of a given number without lookup table:



withoutloup.s

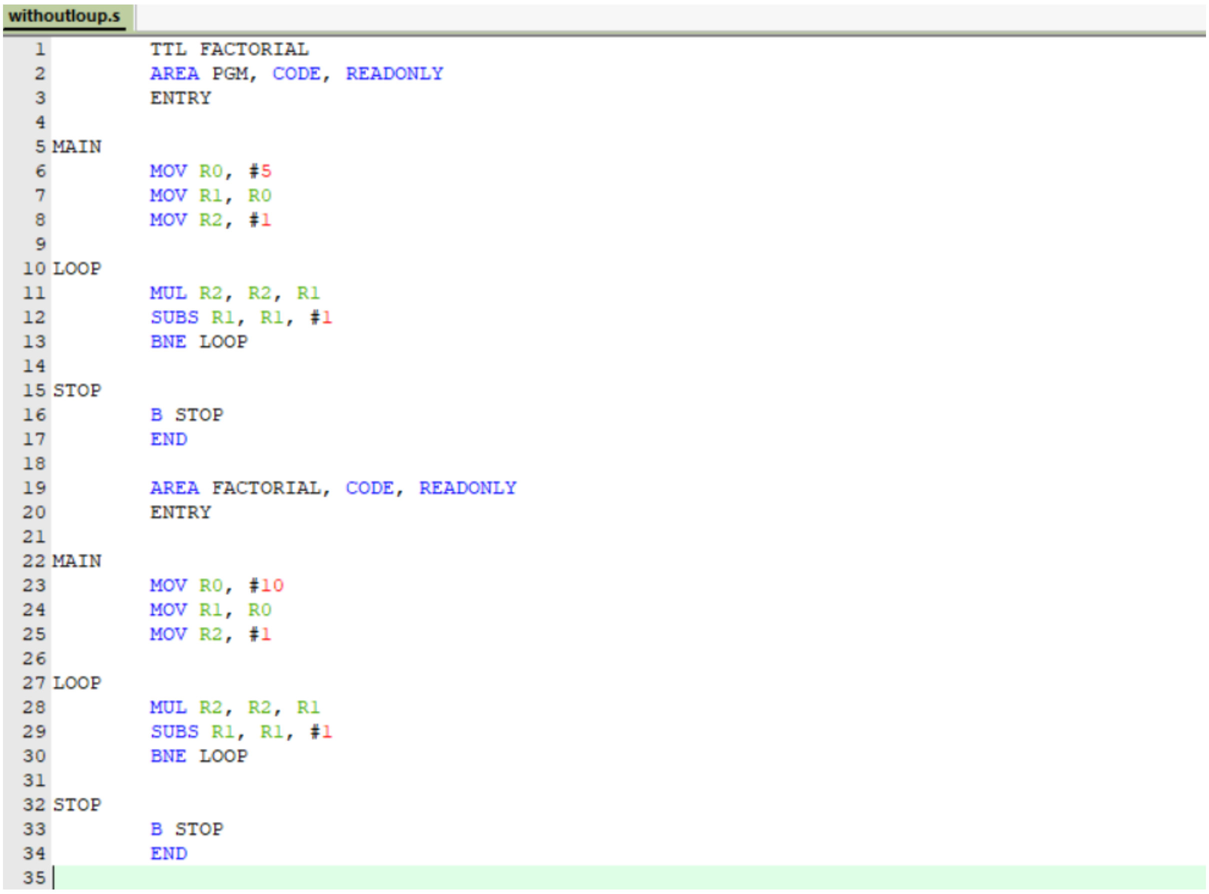
```
TTL FACTORIAL
    AREA PGM, CODE, READONLY
    ENTRY
MAIN
    LDR R0, =NUM
    MOV R1, #1
    MOV R2, #1

LOOP
    MUL R3, R2, R1
    MOV R1, R3
    CMP R2, R0
    BGT ENDFACTOR
    ADD R2, R2, #1
    B LOOP

STOP
    B STOP
    END

AREA FACTORIAL, CODE, READONLY
ENTRY
MAIN
    MOV R0, #10
    MOV R1, R0
    MOV R3, R0

LOOP
    SUBS R1, R1, #1
    MULNE R2, R1, R0
    MOV R0, R2
    BNE LOOP

STOP
    B STOP
    END
```

Output:



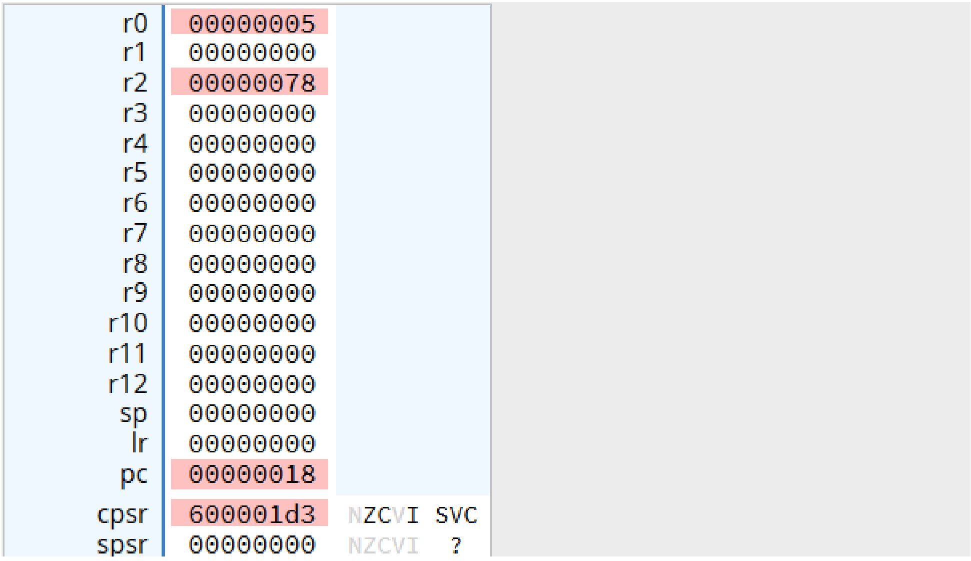
```
  r0  00000005
  r1  000002d0
  r2  00000006
  r3  000002d0
  r4  00000000
  r5  00000000
  r6  00000000
  r7  00000000
  r8  00000000
  r9  00000000
 r10  00000000
 r11  00000000
 r12  00000000
  sp  00000000
  lr  00000000
  pc  00000024

cpsr  200001d3  NZCVI  SVC
spsr  00000000  NZCVI   ?

  s0  00000000
  s1  00000000
  s2  00000000
  s3  00000000
```

Alternate code:



```
        TTL FACTORIAL
        AREA PGM, CODE, READONLY
        ENTRY

MAIN
        MOV R0, #5
        MOV R1, R0
        MOV R2, #1

LOOP
        MUL R2, R2, R1
        SUBS R1, R1, #1
        BNE LOOP

STOP
        B STOP
        END

        AREA FACTORIAL, CODE, READONLY
        ENTRY

MAIN
        MOV R0, #10
        MOV R1, R0
        MOV R2, #1

LOOP
        MUL R2, R2, R1
        SUBS R1, R1, #1
        BNE LOOP

STOP
        B STOP
        END
```

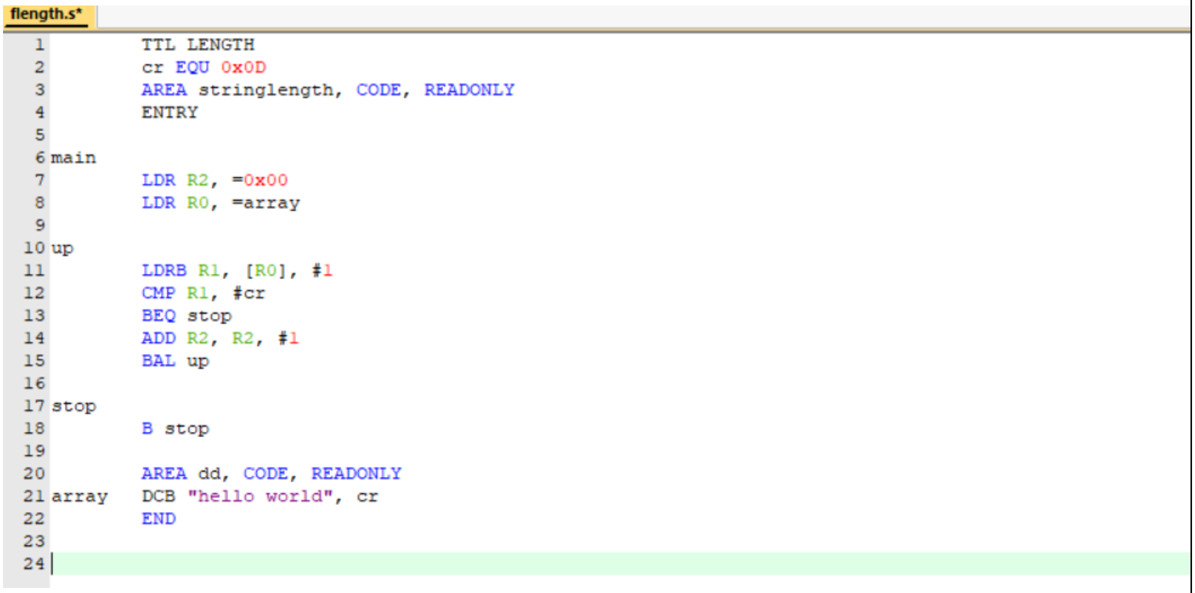
Output:



```
  r0   00000005
  r1   00000000
  r2   00000078
  r3   00000000
  r4   00000000
  r5   00000000
  r6   00000000
  r7   00000000
  r8   00000000
  r9   00000000
 r10   00000000
 r11   00000000
 r12   00000000
  sp   00000000
  lr   00000000
  pc   00000018

cpsr   600001d3   NZCVI SVC
spsr   00000000   NZCVI  ?
```

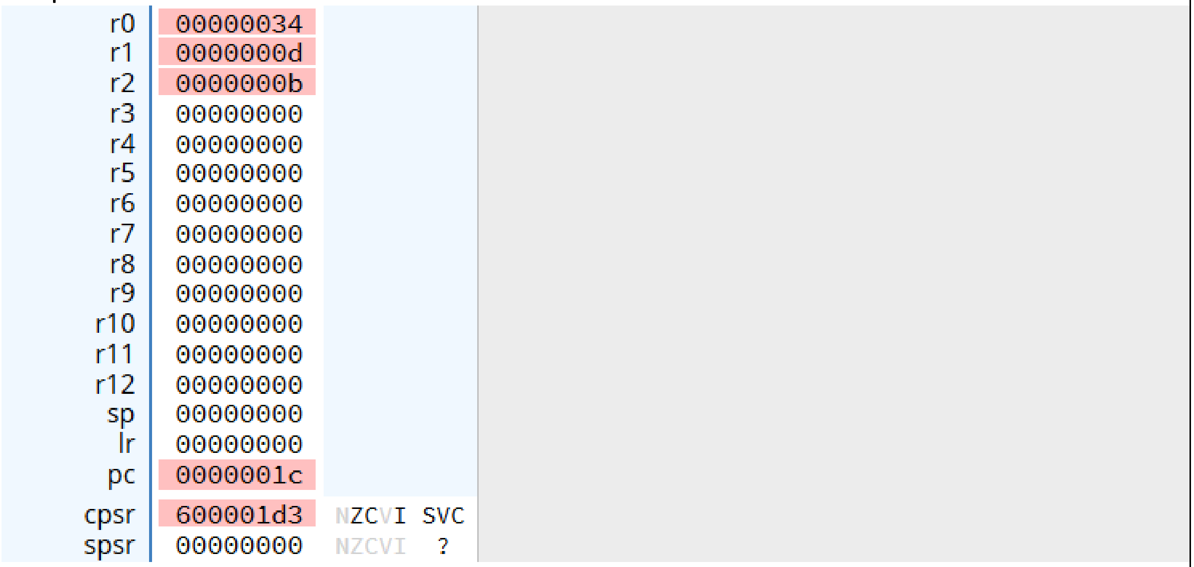
Expt./ Job No.3: Write an ALP to:

i. Find the length of the carriage return terminated string



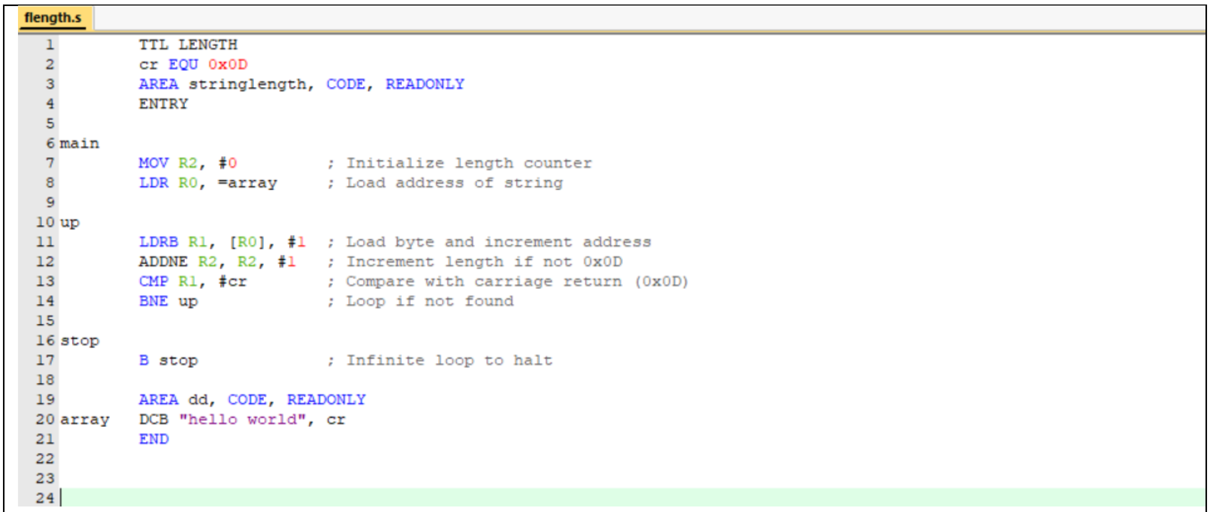
```
        TTL LENGTH
        cr EQU 0x0D
        AREA stringlength, CODE, READONLY
        ENTRY

main
        LDR R2, =0x00
        LDR R0, =array

up
        LDRB R1, [R0], #1
        CMP R1, #cr
        BEQ stop
        ADD R2, R2, #1
        BAL up

stop
        B stop

        AREA dd, CODE, READONLY
array   DCB "hello world", cr
        END

```

Output:



```
  r0   00000034
  r1   0000000d
  r2   0000000b
  r3   00000000
  r4   00000000
  r5   00000000
  r6   00000000
  r7   00000000
  r8   00000000
  r9   00000000
 r10   00000000
 r11   00000000
 r12   00000000
  sp   00000000
  lr   00000000
  pc   0000001c

cpsr   600001d3   NZCVI SVC
spsr   00000000   NZCVI  ?
```

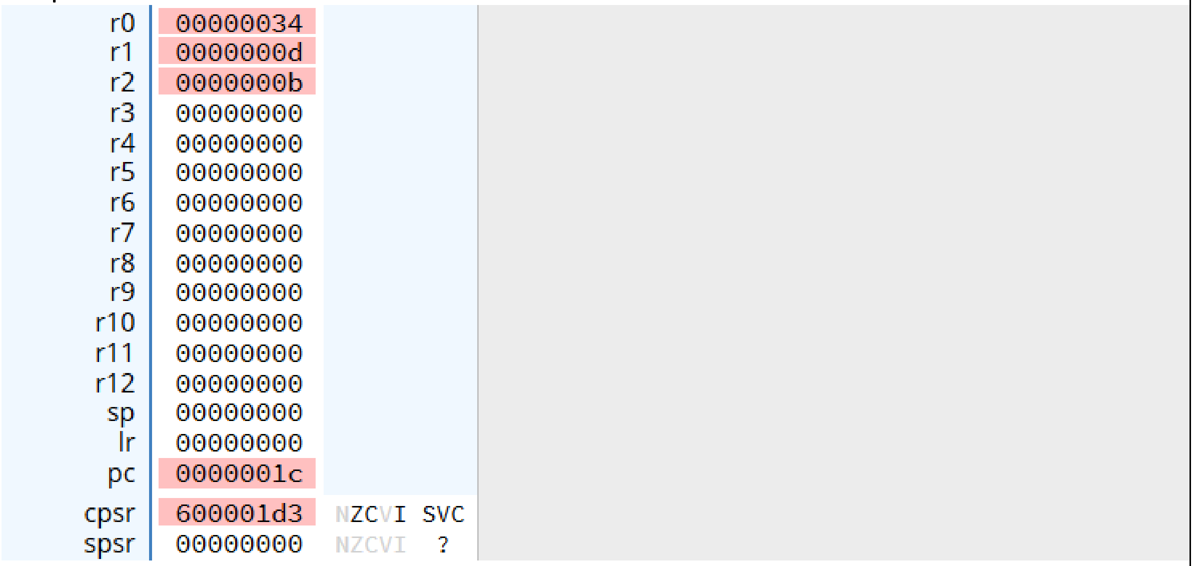
Alternate code:



flength.s

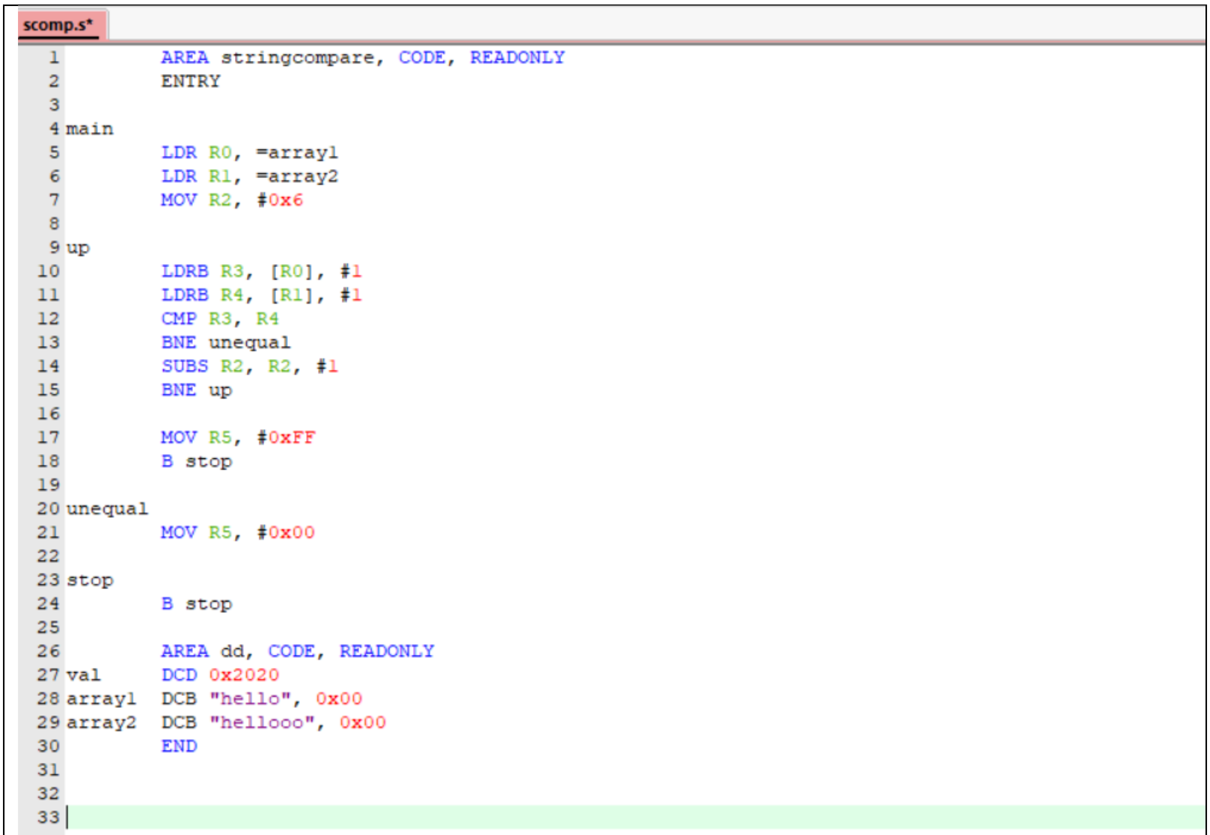
```
         TTL LENGTH
         cr EQU 0x0D
         AREA stringlength, CODE, READONLY
         ENTRY

main
         MOV R2, #0          ; Initialize length counter
         LDR R0, =array      ; Load address of string

up
         LDRB R1, [R0], #1   ; Load byte and increment address
         ADDNE R2, R2, #1    ; Increment length if not 0x0D
         CMP R1, #cr         ; Compare with carriage return (0x0D)
         BNE up              ; Loop if not found

stop
         B stop              ; Infinite loop to halt

         AREA dd, CODE, READONLY
array    DCB "hello world", cr
         END
```

Output:



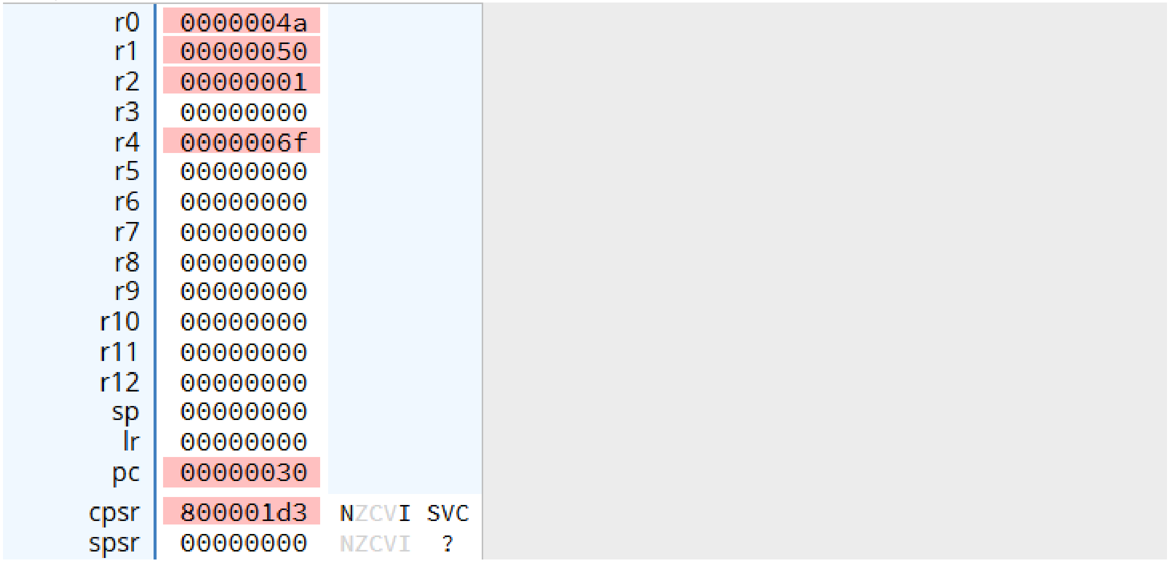
```
  r0  00000034
  r1  0000000d
  r2  0000000b
  r3  00000000
  r4  00000000
  r5  00000000
  r6  00000000
  r7  00000000
  r8  00000000
  r9  00000000
 r10  00000000
 r11  00000000
 r12  00000000
  sp  00000000
  lr  00000000
  pc  0000001c

cpsr  600001d3  NZCVI SVC
spsr  00000000  NZCVI  ?
```

ii.Write an ALP to compare two strings for equality



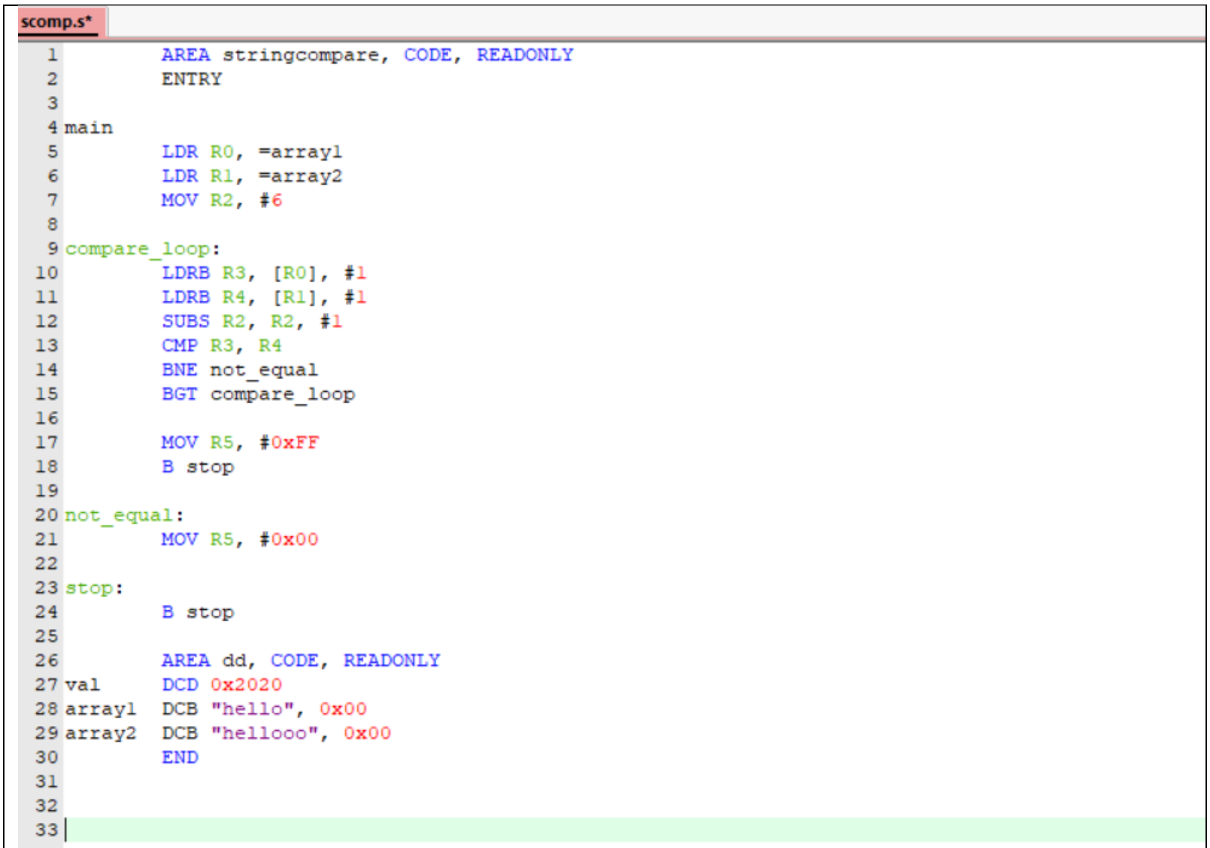
```
        AREA stringcompare, CODE, READONLY
        ENTRY

main
        LDR R0, =array1
        LDR R1, =array2
        MOV R2, #0x6

up
        LDRB R3, [R0], #1
        LDRB R4, [R1], #1
        CMP R3, R4
        BNE unequal
        SUBS R2, R2, #1
        BNE up

        MOV R5, #0xFF
        B stop

unequal
        MOV R5, #0x00

stop
        B stop

        AREA dd, CODE, READONLY
val     DCD 0x2020
array1  DCB "hello", 0x00
array2  DCB "hellooo", 0x00
        END
```

Output:



```
  r0   0000004a
  r1   00000050
  r2   00000001
  r3   00000000
  r4   0000006f
  r5   00000000
  r6   00000000
  r7   00000000
  r8   00000000
  r9   00000000
 r10   00000000
 r11   00000000
 r12   00000000
  sp   00000000
  lr   00000000
  pc   00000030

cpsr   800001d3   NZCVI SVC
spsr   00000000   NZCVI  ?
```

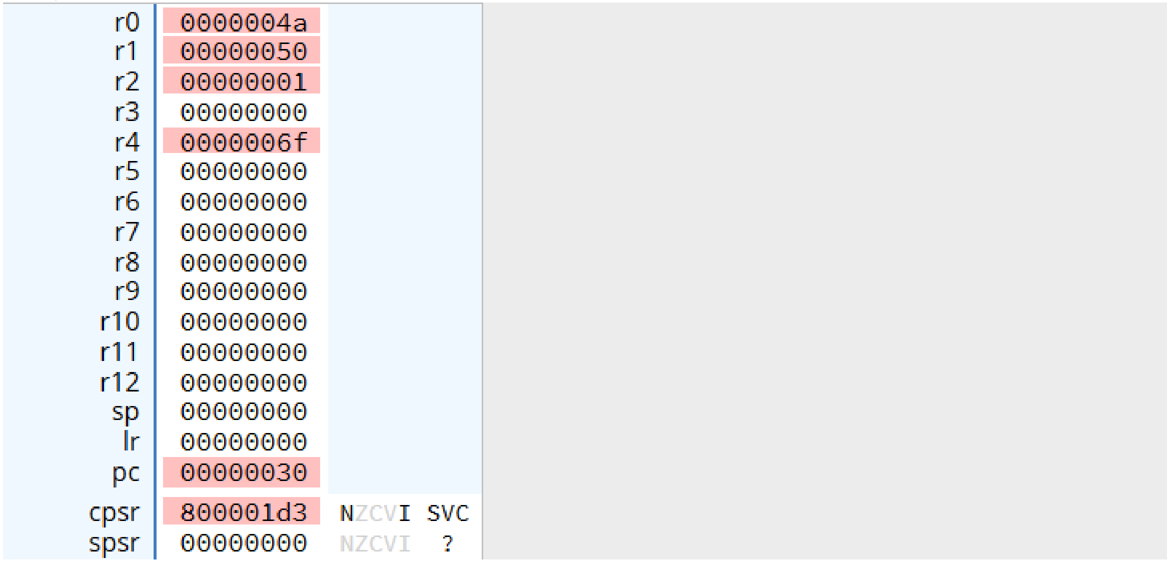
Alternate code:



scomp.s*

```
        AREA stringcompare, CODE, READONLY
        ENTRY

main
        LDR R0, =array1
        LDR R1, =array2
        MOV R2, #6

compare_loop:
        LDRB R3, [R0], #1
        LDRB R4, [R1], #1
        SUBS R2, R2, #1
        CMP R3, R4
        BNE not_equal
        BGT compare_loop

        MOV R5, #0xFF
        B stop

not_equal:
        MOV R5, #0x00

stop:
        B stop

        AREA dd, CODE, READONLY
val     DCD 0x2020
array1  DCB "hello", 0x00
array2  DCB "hellooo", 0x00
        END


```

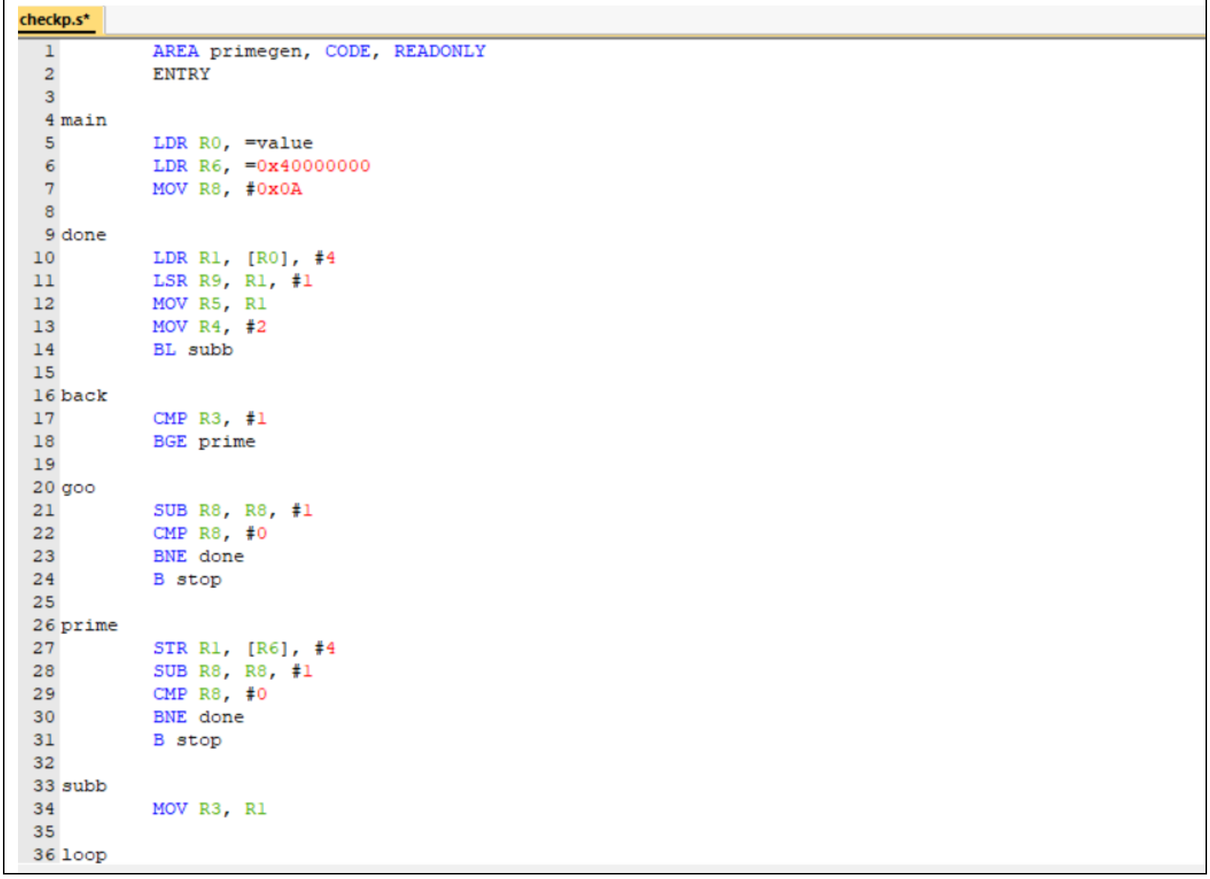
Output:



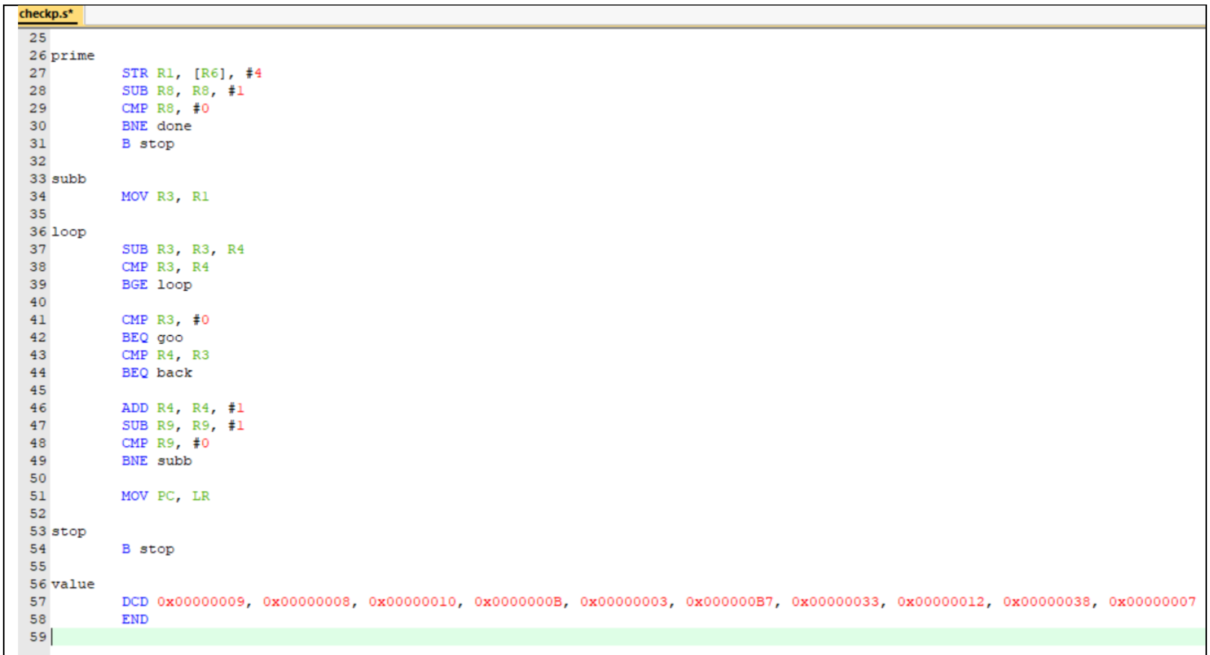
```
  r0   0000004a
  r1   00000050
  r2   00000001
  r3   00000000
  r4   0000006f
  r5   00000000
  r6   00000000
  r7   00000000
  r8   00000000
  r9   00000000
 r10   00000000
 r11   00000000
 r12   00000000
  sp   00000000
  lr   00000000
  pc   00000030

cpsr   800001d3   NZCVI SVC
spsr   00000000   NZCVI  ?
```

4. Write an ALP to pass parameters to a subroutine to find the factorial of a number or prime number generation.

Apply suitable machine dependent optimization technique and analyze for memory and time consumed



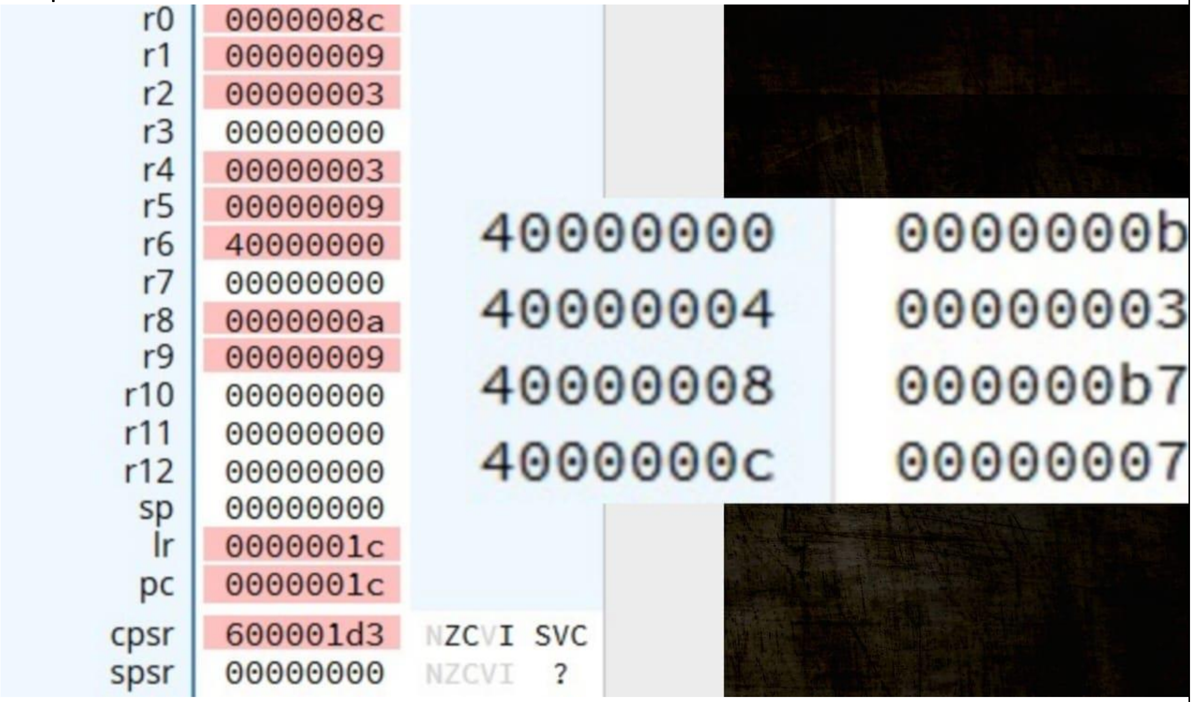
```
        AREA primegen, CODE, READONLY
        ENTRY

main
        LDR R0, =value
        LDR R6, =0x40000000
        MOV R8, #0x0A

done
        LDR R1, [R0], #4
        LSR R9, R1, #1
        MOV R5, R1
        MOV R4, #2
        BL subb

back
        CMP R3, #1
        BGE prime

goo
        SUB R8, R8, #1
        CMP R8, #0
        BNE done
        B stop

prime
        STR R1, [R6], #4
        SUB R8, R8, #1
        CMP R8, #0
        BNE done
        B stop

subb
        MOV R3, R1

loop
```



checkp.s*

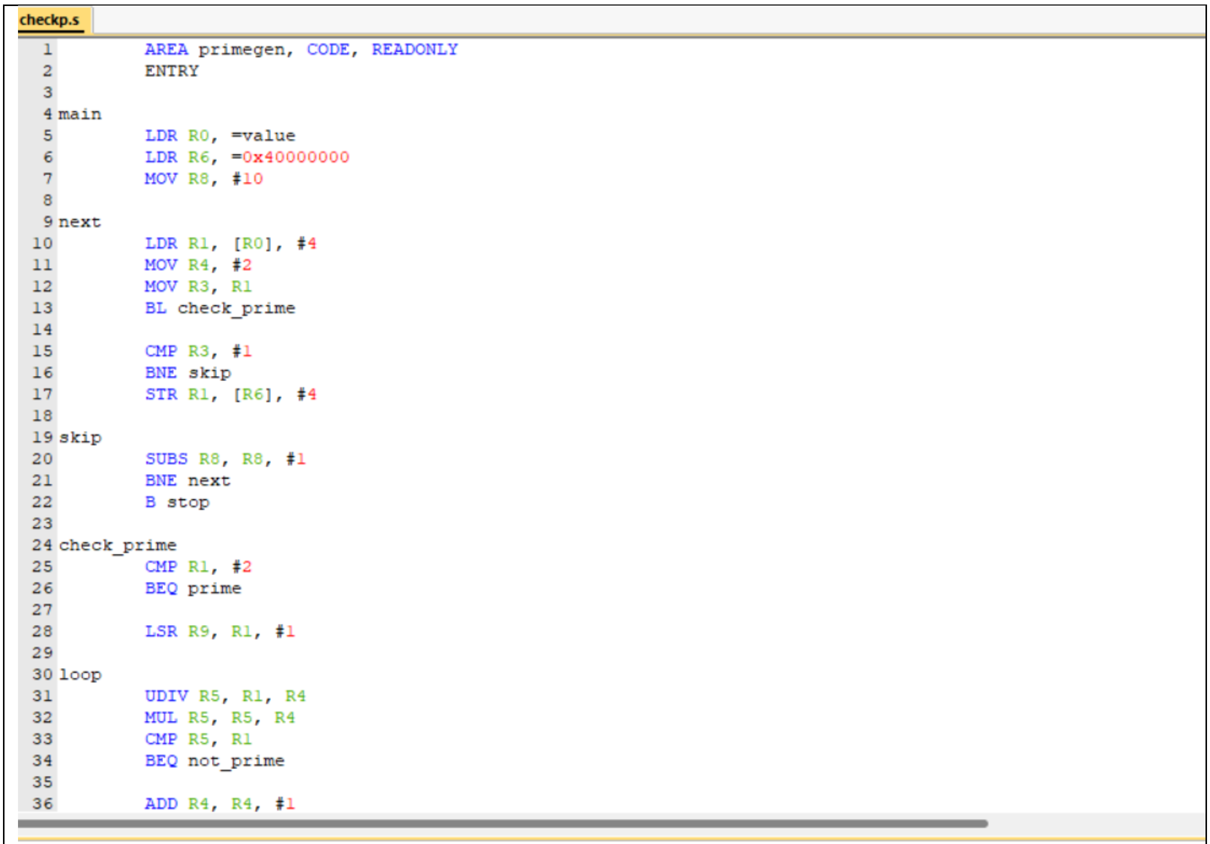
```
prime
        STR R1, [R6], #4
        SUB R8, R8, #1
        CMP R8, #0
        BNE done
        B stop

subb
        MOV R3, R1

loop
        SUB R3, R3, R4
        CMP R3, R4
        BGE loop

        CMP R3, #0
        BEQ goo
        CMP R4, R3
        BEQ back

        ADD R4, R4, #1
        SUB R9, R9, #1
        CMP R9, #0
        BNE subb

        MOV PC, LR

stop
        B stop

value
        DCD 0x00000009, 0x00000008, 0x00000010, 0x0000000B, 0x00000003, 0x000000B7, 0x00000033, 0x00000012, 0x00000038, 0x00000007
        END
```

Output:



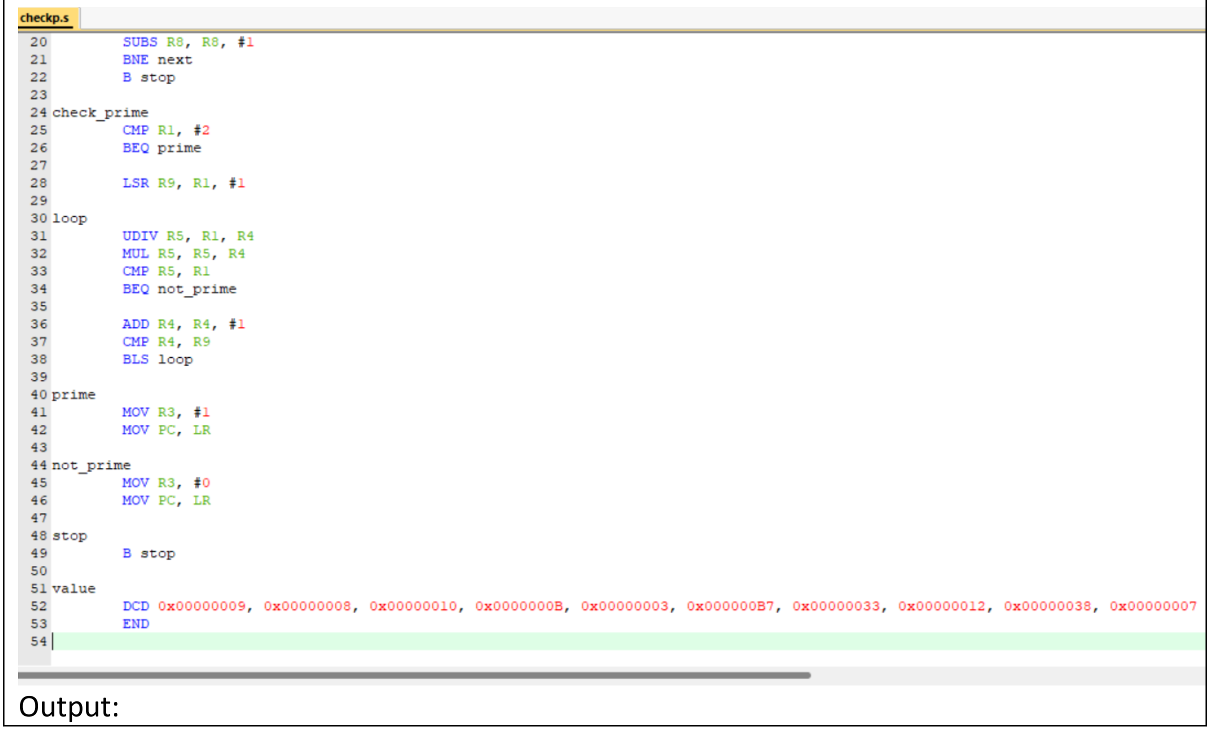
```
  r0   0000008c
  r1   00000009
  r2   00000003
  r3   00000000
  r4   00000003
  r5   00000009     40000000   0000000b
  r6   40000000
  r7   00000000     40000004   00000003
  r8   0000000a
  r9   00000009     40000008   000000b7
 r10   00000000
 r11   00000000     4000000c   00000007
 r12   00000000
  sp   00000000
  lr   0000001c
  pc   0000001c

cpsr   600001d3   NZCVI SVC
spsr   00000000   NZCVI  ?
```

Alternate code:



checkp.s

```
        AREA primegen, CODE, READONLY
        ENTRY

main
        LDR R0, =value
        LDR R6, =0x40000000
        MOV R8, #10

next
        LDR R1, [R0], #4
        MOV R4, #2
        MOV R3, R1
        BL check_prime

        CMP R3, #1
        BNE skip
        STR R1, [R6], #4

skip
        SUBS R8, R8, #1
        BNE next
        B stop

check_prime
        CMP R1, #2
        BEQ prime

        LSR R9, R1, #1

loop
        UDIV R5, R1, R4
        MUL R5, R5, R4
        CMP R5, R1
        BEQ not_prime

        ADD R4, R4, #1
```

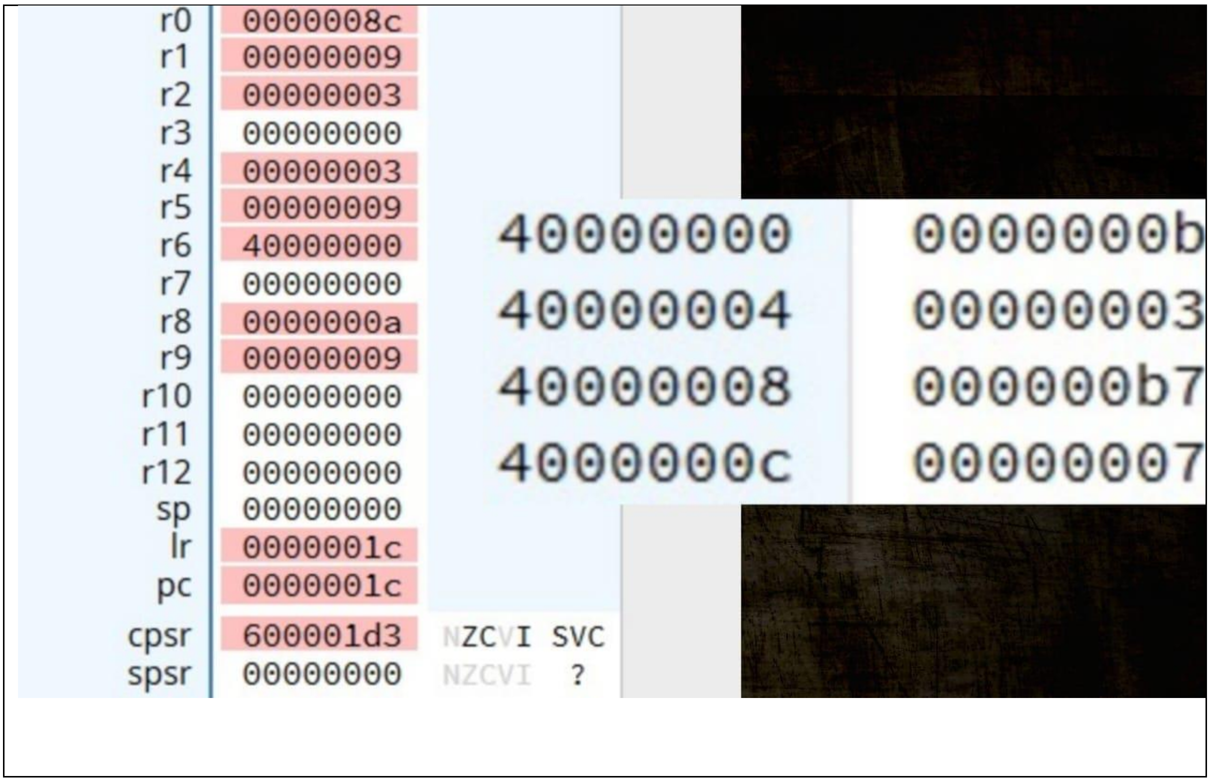


checkp.s

```
        SUBS R8, R8, #1
        BNE next
        B stop

check_prime
        CMP R1, #2
        BEQ prime

        LSR R9, R1, #1

loop
        UDIV R5, R1, R4
        MUL R5, R5, R4
        CMP R5, R1
        BEQ not_prime

        ADD R4, R4, #1
```

```
        CMP R4, R9
        BLS loop

prime
        MOV R3, #1
        MOV PC, LR

not_prime
        MOV R3, #0
        MOV PC, LR

stop
        B stop

value
        DCD 0x00000009, 0x00000008, 0x00000010, 0x0000000B, 0x00000003, 0x000000B7, 0x00000033, 0x00000012, 0x00000038, 0x00000007
        END

```

Output:



```
  r0   0000008c        40000000      0000000b
  r1   00000009        40000004      00000003
  r2   00000003        40000008      000000b7
  r3   00000000        4000000c      00000007
  r4   00000003
  r5   00000009
  r6   40000000
  r7   00000000
  r8   0000000a
  r9   00000009
 r10   00000000
 r11   00000000
 r12   00000000
  sp   00000000
  lr   0000001c
  pc   0000001c
cpsr   600001d3   NZCVI  SVC
spsr   00000000   NZCVI  ?
```

MEMEORY AND TIME

EXPT-1

CODES	TIME	MEMORY
CODE i-a (sub)	0.490 us	
i-b	1.333 us	
CODE ii-a (add)	0.490 us	
ii-b	1.333 us	
CODE iii-a(64bit-add)	0.748 us	
iii- b	1.082 us	
CODE iv-a(mul)	0.833 us	
iv- b	1.417 us	
CODE v-a(div)	5.75 us	
v- b	1.083 us	

EXPT -2

CODES	TIME	MEMORY
CODE i-a(‘N’16bit-sum)	4.833 us	
i-b		
CODE ii-a(max)	8.833 us	
ii-b		
CODE iii-a(min)	7.583 us	
iii-b		