Experiment 1

Aim:

a) Load a value on to R0 and copy it using move command into other general purpose registers.
b) Addition of two values with immediate, direct and indirect mode of addressing

Tool Used:

Keil uVision4

Theory:

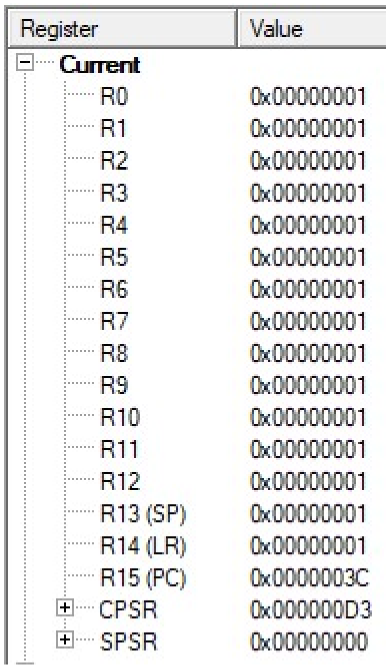
LDR loads the value into the register, MOV copies the value at 2nd register to the 1st register. Here we have loaded the value of R0 with the direct value using = keyword.

Part A:

Code:

```
 AREA PROGRAM, CODE, READONLY
 ENTRY
MAIN
 LDR R0, =0x00000001 ;Load value of R0
 ;copy the value to all other registers
 MOV R1, R0
 MOV R2, R0
 MOV R3, R0
 MOV R4, R0
 MOV R5, R0
 MOV R6, R0
 MOV R7, R0
 MOV R8, R0
 MOV R9, R0
 MOV R10, R0
 MOV R11, R0
 MOV R12, R0
 MOV R13, R0
 MOV R14, R0
 END
```

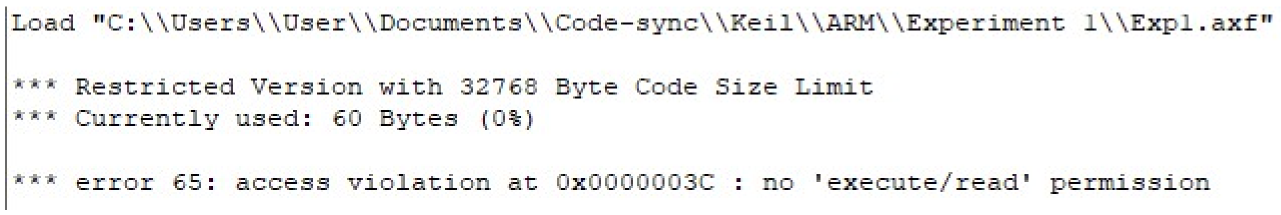
Register Output:



```
Register        Value
Current
  R0            0x00000001
  R1            0x00000001
  R2            0x00000001
  R3            0x00000001
  R4            0x00000001
  R5            0x00000001
  R6            0x00000001
  R7            0x00000001
  R8            0x00000001
  R9            0x00000001
  R10           0x00000001
  R11           0x00000001
  R12           0x00000001
  R13 (SP)      0x00000001
  R14 (LR)      0x00000001
  R15 (PC)      0x0000003C
  CPSR          0x000000D3
  SPSR          0x00000000
```

The program counter results in 3C indicating that 3C/4 = 15 instruction have been performed.

Outputs:



```
Load "C:\\Users\\User\\Documents\\Code-sync\\Keil\\ARM\\Experiment 1\\Exp1.axf"

*** Restricted Version with 32768 Byte Code Size Limit
*** Currently used: 60 Bytes (0%)

*** error 65: access violation at 0x0000003C : no 'execute/read' permission
```

Result:

The value of R0 is copied onto all other registers and is verified to be correct.

Part B

Theory:

Immediate Addressing: Used in cases where we want to load a constant value into an address. Example: MOV R0, #10

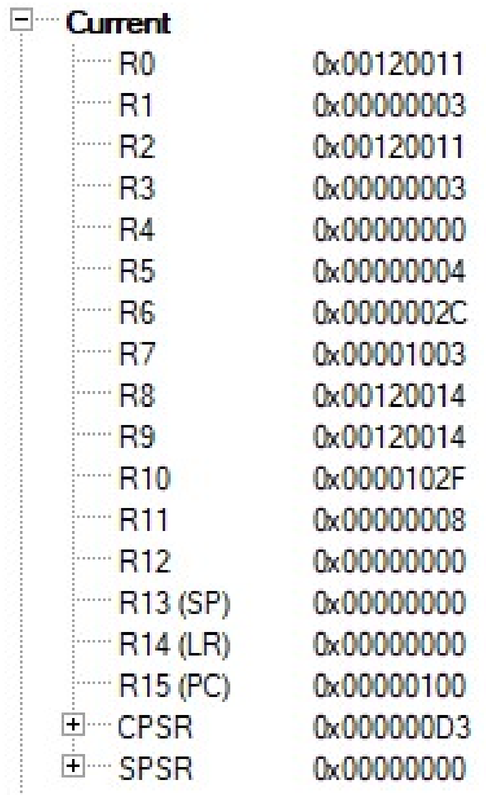
Register Direct Addressing: Used in cases where you want to move data between two registers. Example: MOV R0, R1

Register Indirect Addressing: Used in cases where you want to load data from an address stored in a register. Example: LDR R0, [R1]

Code:

```
AREA PROGRAM, CODE, READONLY
ENTRY
MAIN
 LDR R0, =0x00120011
 LDR R1, =0x00000003
 LDR R2, VALUE1 ;00120011
 LDR R3, VALUE2 ;00000003
 LDR R4, MEM1 ;mem location of R2
 LDR R5, MEM2 ;mem location of R3
 LDR R6, [R4] ;R6 is loaded with data on memory location pointer by R4:=(R2)
 LDR R7, [R5] ;R7 is loaded with data on memory location pointer by R5:=(R3)
 ADD R8, R0, R1 ;immediate addressing
 ADD R9, R2, R3 ;direct addressing
 ADD R10, R6, R7 ;indirect addressing
 LDR R11, =0x00000008 ;mem location to store the result of R11
 STR R10, [R11] ;store value of R10 to mem location pointed by R11
 AREA PROGRAM, DATA, READONLY
VALUE1 DCD &00120011 ; DCD = Define Constant Double word
VALUE2 DCD &00000003
MEM1 DCD &00000000 ;memory address of 1st location
MEM2 DCD &00000004 ;memory address of 2nd location
 END
```

Register Output:



```
Current
  R0        0x00120011
  R1        0x00000003
  R2        0x00120011
  R3        0x00000003
  R4        0x00000000
  R5        0x00000004
  R6        0x0000002C
  R7        0x00001003
  R8        0x00120014
  R9        0x00120014
  R10       0x0000102F
  R11       0x00000008
  R12       0x00000000
  R13 (SP)  0x00000000
  R14 (LR)  0x00000000
  R15 (PC)  0x00000100
+ CPSR      0x000000D3
+ SPSR      0x00000000
```

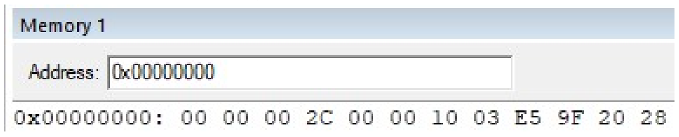
For Inputs at R0 and R1 the input is fed using immediate addressing and the output value is stored at R8.

For Inputs at R2 and R3 the input is loaded using direct addressing and the output is stored at R9.

The value at R6 and R7 is changed at the memory locations and loaded using indirect addressing. The output is stored at R10 and also stored back at memory location using indirect addressing.

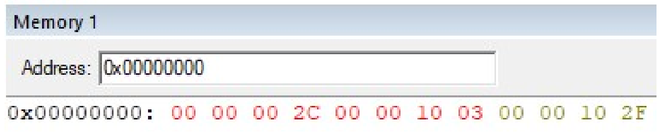
Indirect Addressing Result:

Before Running



```
Memory 1
Address: 0x00000000
0x00000000: 00 00 00 2C 00 00 10 03 E5 9F 20 28
```

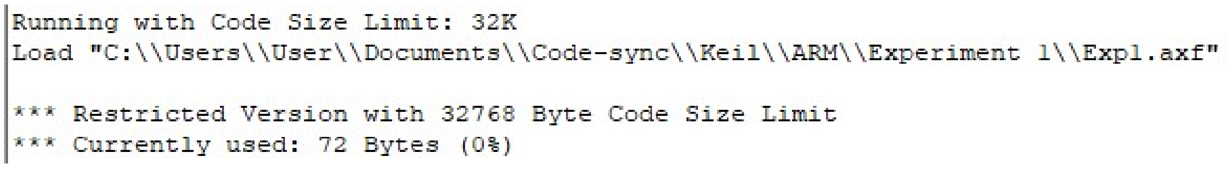
After Running



```
Memory 1
Address: 0x00000000
0x00000000: 00 00 00 2C 00 00 10 03 00 00 10 2F
```

The result 0000102F is visible at the 3rd memory location.

Outputs:



```
Running with Code Size Limit: 32K
Load "C:\\Users\\User\\Documents\\Code-sync\\Keil\\ARM\\Experiment 1\\Expl.axf"

*** Restricted Version with 32768 Byte Code Size Limit
*** Currently used: 72 Bytes (0%)
```

Result:

The values are verified after addition and found to be correct on all the three modes of addressing.

Experiment 2

Aim:

To write an ARM Assembly Language to load any register with 32 bit data and perform the following.

a) Shift left by 2 bit.
b) Shift right by number of bits stored in register.
c) Shift left 5 bits conditionally when ‘0’ flag is set.
d) Arithmetic Shift right by number of bits stored in register.

Tool Used:

Keil uVision4

Theory:

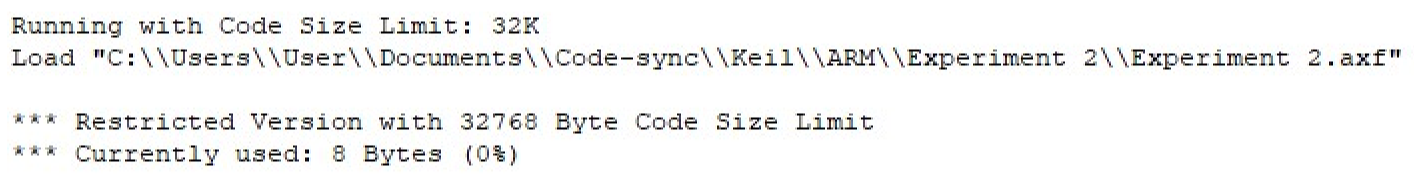
LSL shift the bits left and concatenate a 0 at the LSB. LSR shift the bits right and concatenate a 0 at the MSB. ASR shifts the bits right and concatenate the value of MSB at the new MSB.

a) Shift left by 2 bit.

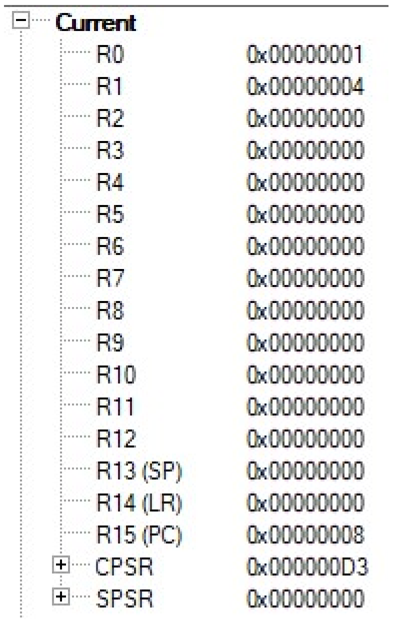
Code:

```
 AREA PROGRAM, CODE, READONLY
 ENTRY
MAIN
 LDR R0,=0x00000001 ;LOAD R0 WITH VALUE 1
 MOV R1,R0,LSL#0x02 ;SHIFT 1 TWO TIMES AN STORE IN R1
 END
```

Output:



```
Running with Code Size Limit: 32K
Load "C:\\Users\\User\\Documents\\Code-sync\\Keil\\ARM\\Experiment 2\\Experiment 2.axf"

*** Restricted Version with 32768 Byte Code Size Limit
*** Currently used: 8 Bytes (0%)
```



```
Current
  R0          0x00000001
  R1          0x00000004
  R2          0x00000000
  R3          0x00000000
  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)    0x00000008
  CPSR        0x000000D3
  SPSR        0x00000000
```

The Value at R0 is 1 and it is shifted 2 times resulting in R1 to be 4.

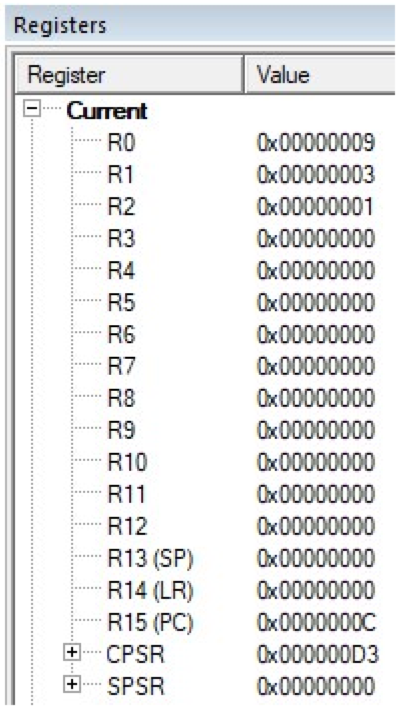
The program Counter has the value 0x00000008 indicating that 2 32-bit instructions have been executed.

b) Shift right by number of bits stored in register.

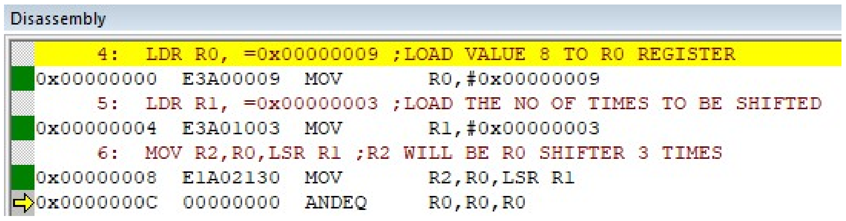
Code:

```
        AREA PROGRAM, CODE, READONLY
        ENTRY
       MAIN
        LDR R0, =0x00000009 ;LOAD VALUE 9 TO R0 REGISTER
        LDR R1, =0x00000003 ;LOAD THE NO OF TIMES TO BE SHIFTED
        MOV R2,R0,LSR R1 ;R2 WILL BE R0 SHIFTER 3 TIMES
        END
```

Output:



```
Registers
Register      Value
Current
  R0          0x00000009
  R1          0x00000003
  R2          0x00000001
  R3          0x00000000
  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
```



```
Disassembly
     4:  LDR R0, =0x00000009 ;LOAD VALUE 8 TO R0 REGISTER
0x00000000  E3A00009  MOV        R0,#0x00000009
     5:  LDR R1, =0x00000003 ;LOAD THE NO OF TIMES TO BE SHIFTED
0x00000004  E3A01003  MOV        R1,#0x00000003
     6:  MOV R2,R0,LSR R1 ;R2 WILL BE R0 SHIFTER 3 TIMES
0x00000008  E1A02130  MOV        R2,R0,LSR R1
0x0000000C  00000000  ANDEQ      R0,R0,R0
```

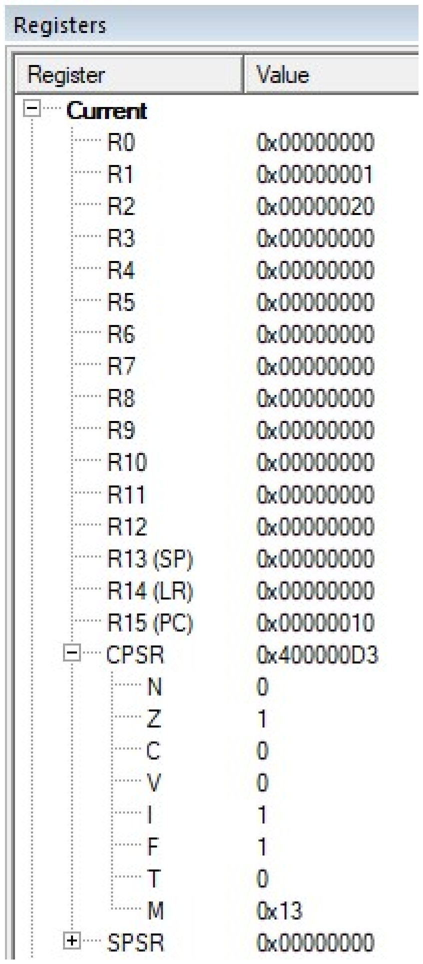
The value stored in R0 is 9 and it is shifted right 3 times resulting in a value of 1 at R2 where the lower bits are truncated.

c) Shift left 5 bits conditionally when ‘0’ flag is set.

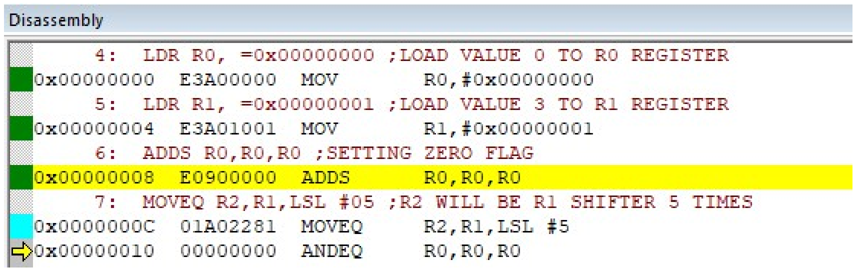
Code:

```
      AREA PROGRAM, CODE, READONLY
      ENTRY
     MAIN
      LDR R0, =0x00000000 ;LOAD VALUE 0 TO R0 REGISTER
      LDR R1, =0x00000001 ;LOAD VALUE 3 TO R1 REGISTER
      ADDS R0,R0,R0 ;SETTING ZERO FLAG
      MOVEQ R2,R1,LSL #05 ;R2 WILL BE R1 SHIFTER 5 TIMES
      END
```

Output:



```
Registers
Register        Value
Current
  R0            0x00000000
  R1            0x00000001
  R2            0x00000020
  R3            0x00000000
  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)      0x00000010
  CPSR          0x400000D3
    N           0
    Z           1
    C           0
    V           0
    I           1
    F           1
    T           0
    M           0x13
  SPSR          0x00000000
```



```
Disassembly
     4:  LDR R0, =0x00000000 ;LOAD VALUE 0 TO R0 REGISTER
0x00000000  E3A00000  MOV       R0,#0x00000000
     5:  LDR R1, =0x00000001 ;LOAD VALUE 3 TO R1 REGISTER
0x00000004  E3A01001  MOV       R1,#0x00000001
     6:  ADDS R0,R0,R0 ;SETTING ZERO FLAG
0x00000008  E0900000  ADDS      R0,R0,R0
     7:  MOVEQ R2,R1,LSL #05 ;R2 WILL BE R1 SHIFTER 5 TIMES
0x0000000C  01A02281  MOVEQ     R2,R1,LSL #5
0x00000010  00000000  ANDEQ     R0,R0,R0
```

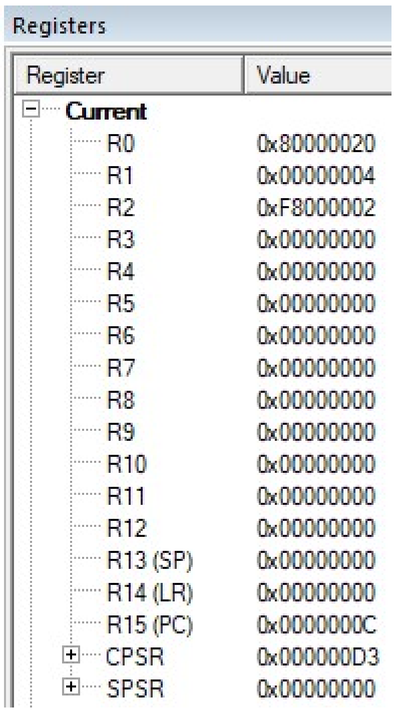
The value stored in R1 is 1 and it is shifted 5 times resulting in a value of 20 at R2 as the Zero flag in CPSR is set to zero.

d) Arithmetic Shift right by number of bits stored in register.

Code:

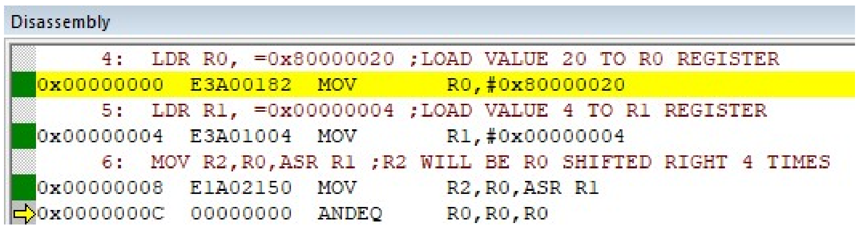
```
       AREA PROGRAM, CODE, READONLY
       ENTRY
      MAIN
       LDR R0, =0x80000020 ;LOAD VALUE WITH MSB 1 TO R0 REGISTER
       LDR R1, =0x00000004 ;LOAD VALUE 4 TO R1 REGISTER
       MOV R2,R0,ASR R1 ;R2 WILL BE R0 SHIFTED RIGHT 4 TIMES
       END
```

Output:



Registers

Register	Value
Current	
R0	0x80000020
R1	0x00000004
R2	0xF8000002
R3	0x00000000
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



Disassembly

```
     4:  LDR R0, =0x80000020 ;LOAD VALUE 20 TO R0 REGISTER
0x00000000  E3A00182  MOV      R0,#0x80000020
     5:  LDR R1, =0x00000004 ;LOAD VALUE 4 TO R1 REGISTER
0x00000004  E3A01004  MOV      R1,#0x00000004
     6:  MOV R2,R0,ASR R1 ;R2 WILL BE R0 SHIFTED RIGHT 4 TIMES
0x00000008  E1A02150  MOV      R2,R0,ASR R1
0x0000000C  00000000  ANDEQ    R0,R0,R0
```

R0 has MSB as 1000_0000 which is shifted right 4 times resulting in 1111_1000(F8) in R2.

Result:

The experiments on shift operations have been performed and verified to be correct.

Experiment 3

Aim:

To write an ARM Assembly Language to copy words from consecutive location and store it in consecutive destination locations.

a) Multiple register transfer instructions.

b) Load and Store instruction in a loop.

Tool Used:

Keil uVision4

Theory:

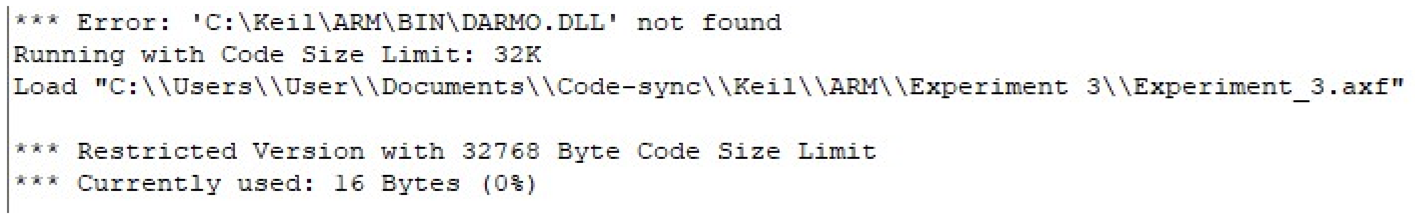
LDM load multiple register locations with starting address mentioned. ! is used in LDM for updating pointer, else same value will be updated in all registers. STM load the value into consecutive memory locations with starting address mentioned.

a) Multiple register transfer instructions.

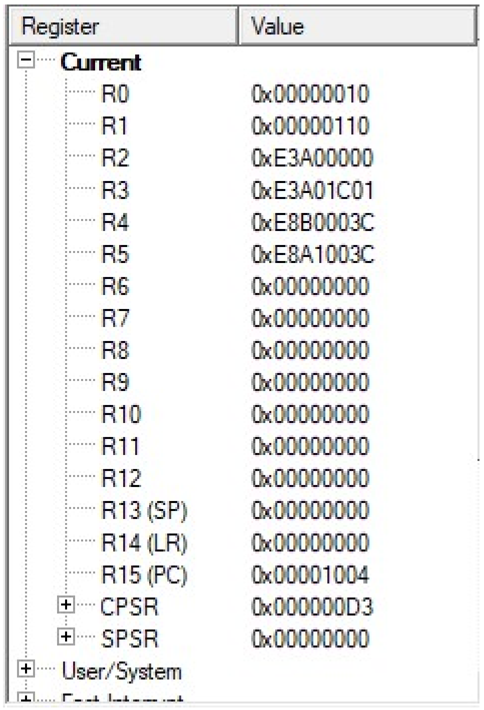
Code:

```
 AREA PROGRAM, CODE, READONLY
 ENTRY
MAIN
 LDR R0, =0x00000000; SOURCE MEMORY LOCATION
 LDR R1, =0x00000100; DESTINATION MEMORY LOCATION
 LDM R0!, {R2-R5}; COPIES CONTINUOSLY FROM EXTERNAL TO R2 TO R5
 STM R1!, {R2-R5}; COPIES CONTINUOSLY FROM R2 TO R5 TO DESTINATION
 END
```

Output:

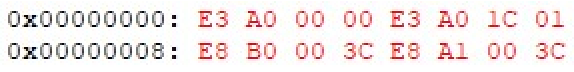


```
*** Error: 'C:\Keil\ARM\BIN\DARMO.DLL' not found
Running with Code Size Limit: 32K
Load "C:\\Users\\User\\Documents\\Code-sync\\Keil\\ARM\\Experiment 3\\Experiment_3.axf"

*** Restricted Version with 32768 Byte Code Size Limit
*** Currently used: 16 Bytes (0%)
```



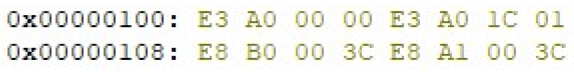
Register	Value
Current	
R0	0x00000010
R1	0x00000110
R2	0xE3A00000
R3	0xE3A01C01
R4	0xE8B0003C
R5	0xE8A1003C
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00001004
CPSR	0x000000D3
SPSR	0x00000000
User/System	

The input data to be copied



```
0x00000000: E3 A0 00 00 E3 A0 1C 01
0x00000008: E8 B0 00 3C E8 A1 00 3C
```

The data copied after running



```
0x00000100: E3 A0 00 00 E3 A0 1C 01
0x00000108: E8 B0 00 3C E8 A1 00 3C
```

4 words starting at 0x…000 is copied at destination address 0x…100.

Using IA, IB, DA, DB

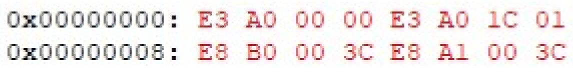
Code:

```
        AREA PROGRAM, CODE, READONLY
        ENTRY
       MAIN
        LDR R0, =0x00000000; SOURCE MEMORY LOCATION
        LDR R1, =0x00000100; DESTINATION MEMORY LOCATION
        LDMIA R0!, {R2-R5}; COPIES CONTINUOSLY FROM EXTERNAL TO R2 TO R5
        STMIA R1!, {R2-R5}; COPIES CONTINUOSLY FROM R2 TO R5 TO DESTINATION
        END
```

Output:

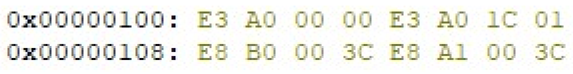
For Increment after [IA]:

Input at 0x00000000



```
0x00000000: E3 A0 00 00 E3 A0 1C 01
0x00000008: E8 B0 00 3C E8 A1 00 3C
```

Input at 0x00000000



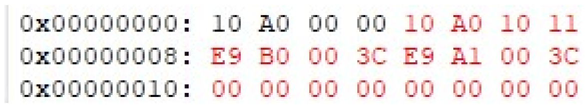
```
0x00000100: E3 A0 00 00 E3 A0 1C 01
0x00000108: E8 B0 00 3C E8 A1 00 3C
```

Value starts at 00 and 100 indicating that the increment is done after copying

For Increment before [IB]:

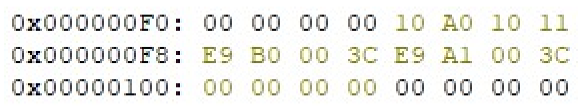
```
LDR R0, =0x00000000; SOURCE MEMORY LOCATION
LDR R1, =0x000000F0; DESTINATION MEMORY LOCATION
LDMIB R0!, {R2-R5}; COPIES CONTINUOSLY FROM EXTERNAL TO R2 TO R5
STMIB R1!, {R2-R5}; COPIES CONTINUOSLY FROM R2 TO R5 TO DESTINATION
```

Input at 0x00000000



```
0x00000000: 10 A0 00 00 10 A0 10 11
0x00000008: E9 B0 00 3C E9 A1 00 3C
0x00000010: 00 00 00 00 00 00 00 00
```

Output at 0x00000000



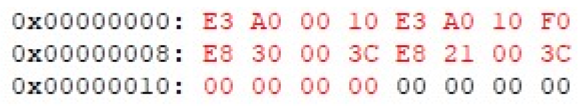
```
0x000000F0: 00 00 00 00 10 A0 10 11
0x000000F8: E9 B0 00 3C E9 A1 00 3C
0x00000100: 00 00 00 00 00 00 00 00
```

Value starts at 04 and F4 instead of F0 indicating that the increment is done before copying

For Decrement After [DA]:

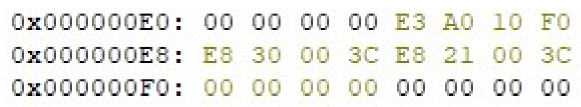
```
LDMDA R0!, {R2-R5}; COPIES CONTINUOSLY FROM EXTERNAL TO R2 TO R5
STMDA R1!, {R2-R5}; COPIES CONTINUOSLY FROM R2 TO R5 TO DESTINATION
```

Input at 0x00000000



```
0x00000000: E3 A0 00 10 E3 A0 10 F0
0x00000008: E8 30 00 3C E8 21 00 3C
0x00000010: 00 00 00 00 00 00 00 00
```

Output at 0x00000000



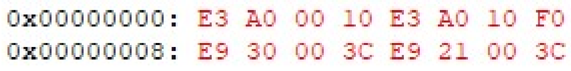
```
0x000000E0: 00 00 00 00 E3 A0 10 F0
0x000000E8: E8 30 00 3C E8 21 00 3C
0x000000F0: 00 00 00 00 00 00 00 00
```

Value starts at 10 and F0 indicating that the decrement is done after copying

For Decrement Before [DB]:

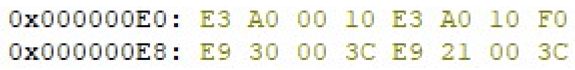
```
        LDMDB R0!, {R2-R5}; COPIES CONTINUOSLY FROM EXTERNAL TO R2 TO R5
        STMDB R1!, {R2-R5}; COPIES CONTINUOSLY FROM R2 TO R5 TO DESTINATION
```

Input at 0x00000000



```
0x00000000: E3 A0 00 10 E3 A0 10 F0
0x00000008: E9 30 00 3C E9 21 00 3C
```

Output at 0x000000E8



```
0x000000E0: E3 A0 00 10 E3 A0 10 F0
0x000000E8: E9 30 00 3C E9 21 00 3C
```

Value starts at 00 and ends at EC indicating that the decrement is done before copying.

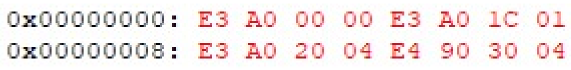
b) Load and Store Instructions in a loop

Code:

```
        AREA PROGRAM, CODE,READONLY
        ENTRY
MAIN
        LDR R0, =0x00000000; SOURCE MEMORY LOCATION
        LDR R1, =0x00000100; DESTINATION MEMORY LOCATION
        MOV R2, #4; 4 LOCATIONS FOR 4 WORDS
FOR     LDR R3, [R0], #4; MOVE THE RESPECTIVE VALUE TO R3
        STR R3, [R1], #4; STORE R3 TO THE RESPECTIVE LOCATION
        SUBS R2,R2,#1; DECREMENT THE COUNTER
        BNE FOR; IF NOT ZERO REPEAT THE LOOP
        END
```

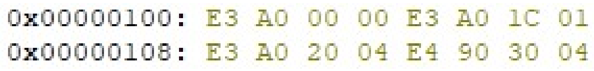
Output:

Before Execution Input at 0x00000000



```
0x00000000: E3 A0 00 00 E3 A0 1C 01
0x00000008: E3 A0 20 04 E4 90 30 04
```

After Execution Output at 0x00000100



```
0x00000100: E3 A0 00 00 E3 A0 1C 01
0x00000108: E3 A0 20 04 E4 90 30 04
```

Result:

The experiments on shift operations have been performed and verified to be correct.

Experiment 4

Aim:

To write an ARM Assembly Language to

a) Add two 64 bit numbers.

b) Add ten 32 bit numbers.

Tool Used:

Keil uVision4

Theory:

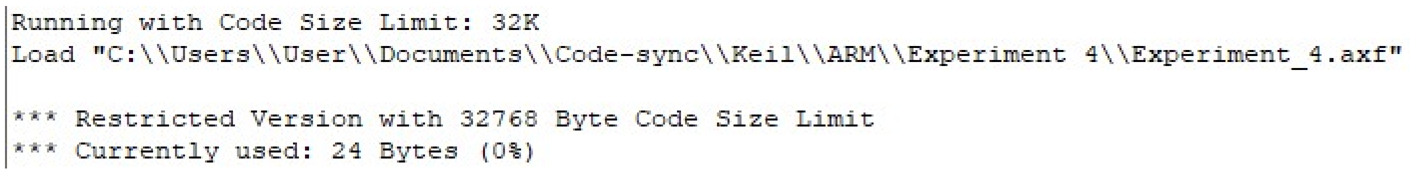
LDM load multiple register locations with starting address mentioned. ! is used in LDM for updating pointer, else same value will be updated in all registers. STM load the value into consecutive memory locations with starting address mentioned. ADDCS adds the value if the carry flag is set.

a) Add two 64 bit numbers.

Code:

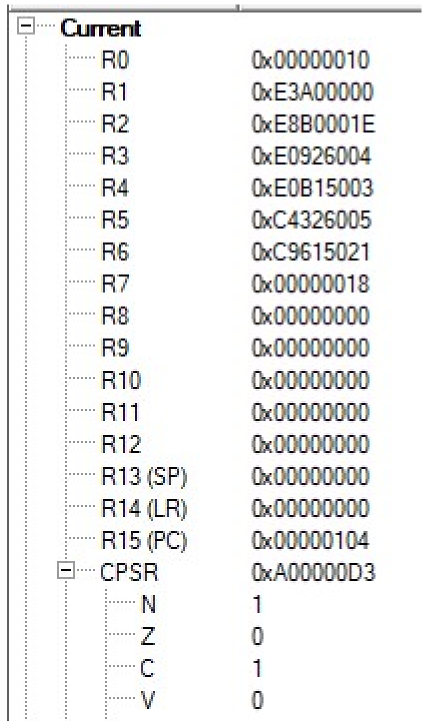
```
 AREA PROGRAM,CODE, READONLY
 ENTRY
MAIN
 LDR R0, =0X00000000; SOURCE MEMORY LOCATION
 LDM R0!, {R1-R4}; COPY TO REGISTERS R1 TO R4 FROM MEM LOCATIONS
 ADDS R6,R2,R4; ADDS THE LOWER NIBBLE
 ADCS R5,R1,R3; ADDS THE UPPER NIBBLE
 LDR R7, =0X00000010; DESTINATION MEMORY LOCATION
 STM R7!, {R5-R6}; STORE RESULTS IN DESTINATION
 END
```

Output:

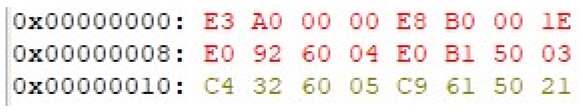


```
Running with Code Size Limit: 32K
Load "C:\\Users\\User\\Documents\\Code-sync\\Keil\\ARM\\Experiment 4\\Experiment_4.axf"

*** Restricted Version with 32768 Byte Code Size Limit
*** Currently used: 24 Bytes (0%)
```

Register Contents


```
Current
  R0        0x00000010
  R1        0xE3A00000
  R2        0xE8B0001E
  R3        0xE0926004
  R4        0xE0B15003
  R5        0xC4326005
  R6        0xC9615021
  R7        0x00000018
  R8        0x00000000
  R9        0x00000000
  R10       0x00000000
  R11       0x00000000
  R12       0x00000000
  R13 (SP)  0x00000000
  R14 (LR)  0x00000000
  R15 (PC)  0x00000104
  CPSR      0xA00000D3
    N       1
    Z       0
    C       1
    V       0
```

The memory location input data and the added value.


```
0x00000000: E3 A0 00 00 E8 B0 00 1E
0x00000008: E0 92 60 04 E0 B1 50 03
0x00000010: C4 32 60 05 C9 61 50 21
```

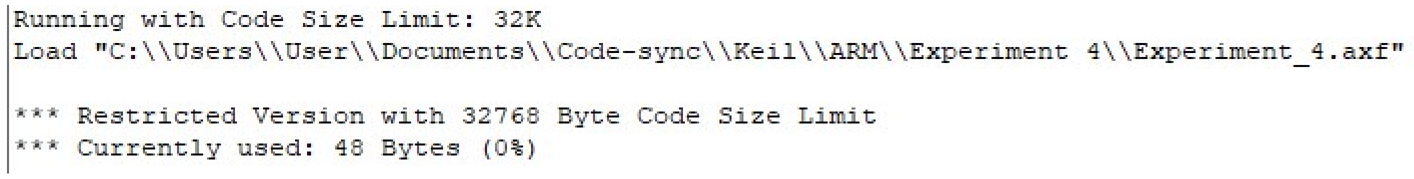
The input starts from 0x…0 to 0x0…08, the output is displayed from 0x00000010

b) Add ten 32 bit numbers.

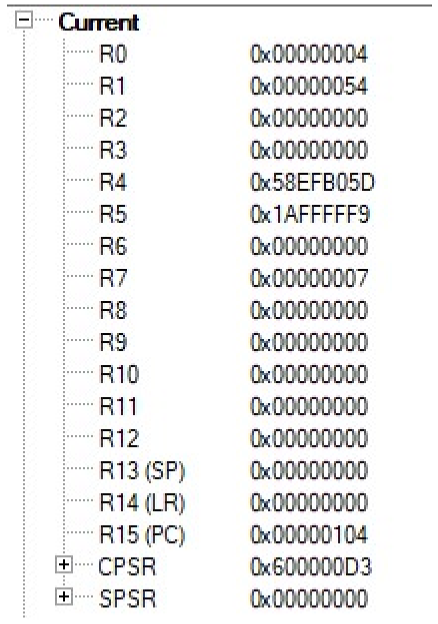
Code:

```
             AREA PROGRAM,CODE, READONLY
            ENTRY
           MAIN
               LDR R0, =0x00000000; SOURCE MEMORY LOCATION
               LDR R1, =0x00000050; DESTINATION MEMORY LOCATION
               MOV R3, #9; COUNTER WITH DATA 10
               LDR R4, [R0]; 1ST VALUE
           FOR ADD R0, R0, #4; INCREMENTED ADDRESS
               LDR R5, [R0]; FURTHER VALUES
               ADDS R4,R4,R5; ADD CONSECUTIVE VALUES
               ADDCS R7,R7,#1; CARRY COUNT
               SUBS R3,R3,#1; DECREMENT THE COUNTER
               BNE FOR; IF NOT ZERO REPEAT THE LOOP
               STR R4, [R1], #4; STORE THE ADDED VALUE
               STR R7, [R1]; STORE NUMBER OF CARRYS IN NEXT LOCATION
               END
```

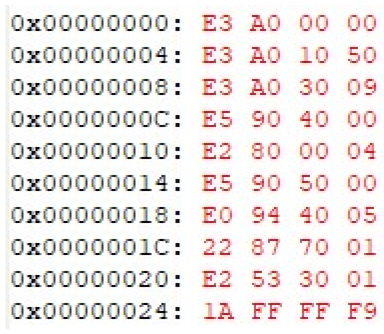
Output:


```
Running with Code Size Limit: 32K
Load "C:\\Users\\User\\Documents\\Code-sync\\Keil\\ARM\\Experiment 4\\Experiment_4.axf"

*** Restricted Version with 32768 Byte Code Size Limit
*** Currently used: 48 Bytes (0%)
```

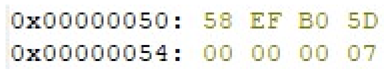
Register Contents


```
Current
R0          0x00000004
R1          0x00000054
R2          0x00000000
R3          0x00000000
R4          0x58EFB05D
R5          0x1AFFFFF9
R6          0x00000000
R7          0x00000007
R8          0x00000000
R9          0x00000000
R10         0x00000000
R11         0x00000000
R12         0x00000000
R13 (SP)    0x00000000
R14 (LR)    0x00000000
R15 (PC)    0x00000104
CPSR        0x600000D3
SPSR        0x00000000
```

The 10 input words


```
0x00000000: E3 A0 00 00
0x00000004: E3 A0 10 50
0x00000008: E3 A0 30 09
0x0000000C: E5 90 40 00
0x00000010: E2 80 00 04
0x00000014: E5 90 50 00
0x00000018: E0 94 40 05
0x0000001C: 22 87 70 01
0x00000020: E2 53 30 01
0x00000024: 1A FF FF F9
```

The sum and carry


```
0x00000050: 58 EF B0 5D
0x00000054: 00 00 00 07
```

The input words range from 0x0…0 to 0x0…024 and the sum is displayed at 0x0…050 and the carry is in 0x0…054.

Result:

The experiments on add operations have been performed and verified to be correct.

Experiment 5

Aim:

To write an ARM Assembly Language to find the number of bytes in a set of 10 locations that match the value 0xAC

Tool Used:

Keil uVision4

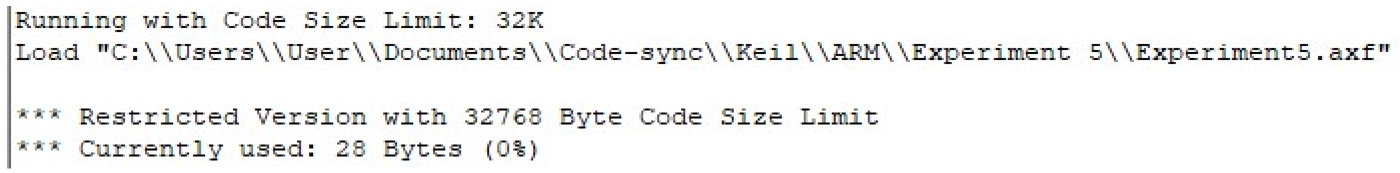
Theory:

LDRB is used to copy just 1 Byte of data to the lower location of the register. CMP compares two operands and if zero sets the zero flag. The EQ condition checks the zero flag for set to let the process happen.

Code:

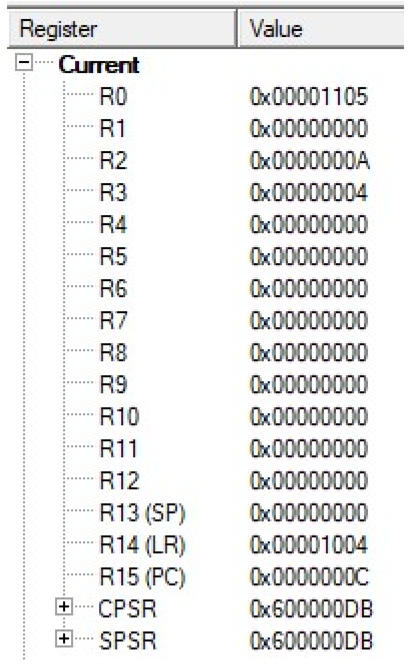
```
 AREA PROGRAM, CODE, READONLY
ENTRY
MAIN
        LDR R0, =0X00001000 //starting location
        MOV R2, #10 // counter for 10 locations
LOOP    LDRB R1, [R0], #1 // load the value
        CMP R1, #0XAC // check if same
        ADDEQ R3,R3,#1 //if equal label
        SUBS R2,R2,#1 // decrement counter
        BNE LOOP // run 10 times
 END
```

Output:



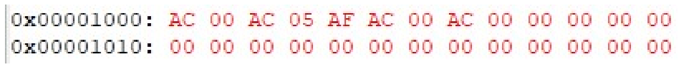
```
Running with Code Size Limit: 32K
Load "C:\\Users\\User\\Documents\\Code-sync\\Keil\\ARM\\Experiment 5\\Experiment5.axf"

*** Restricted Version with 32768 Byte Code Size Limit
*** Currently used: 28 Bytes (0%)
```

Register Contents



Register	Value
Current	
R0	0x00001105
R1	0x00000000
R2	0x0000000A
R3	0x00000004
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00001004
R15 (PC)	0x0000000C
CPSR	0x600000DB
SPSR	0x600000DB

The memory location of input data.



```
0x00001000: AC 00 AC 05 AF AC 00 AC 00 00 00 00 00
0x00001010: 00 00 00 00 00 00 00 00 00 00 00 00 00
```

The input starts from 0x00001000 to 0x0000100A. Output is at R3.

Result:

The experiments on compare operation has been performed and verified to be correct.

Experiment 6

Aim:

To write an ARM Assembly Language to find the number of 1’s and 0’s in a given word.

Tool Used:

Keil uVision4

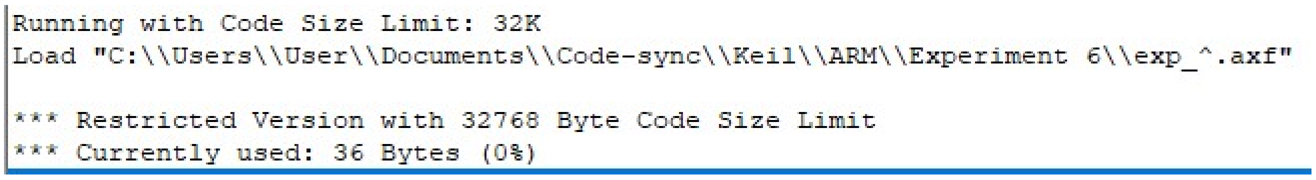
Theory:

RRX rotates the value of the register and store the left most bit to carry, and bring the carry bit if appended at the right most bit.

Code:

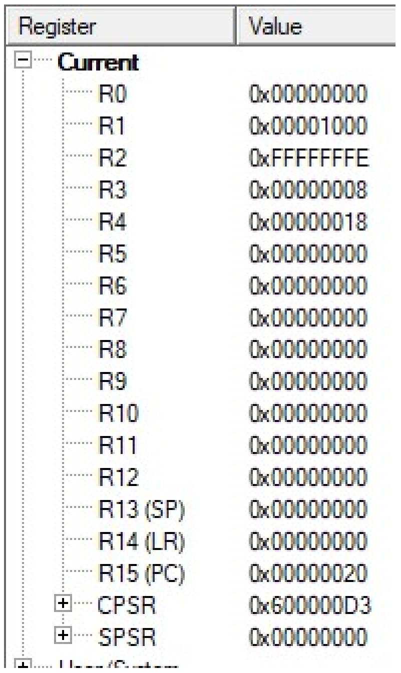
```
     AREA PROGRAM, CODE, READONLY
    ENTRY
   MAIN
          MOV R0, #32 // counter value
          LDR R1, =0x00001000 // input location
          LDR R2, [R1] // load the input at the register
   LOOP   MOVS R2,R2,RRX //rotate the value of reg
          ADDCS R3,R3,#1 // increment if bit is set
          SUBS R0,R0,#1 // decrement counter
          BNE LOOP // loop branch
          RSB R4,R3,#32 // count no of 0's
    END
```

Output:



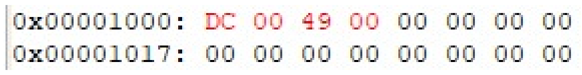
```
Running with Code Size Limit: 32K
Load "C:\\Users\\User\\Documents\\Code-sync\\Keil\\ARM\\Experiment 6\\exp_^.axf"

*** Restricted Version with 32768 Byte Code Size Limit
*** Currently used: 36 Bytes (0%)
```

Register Contents



Register	Value
Current	
R0	0x00000000
R1	0x00001000
R2	0xFFFFFFFE
R3	0x00000008
R4	0x00000018
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	0x600000D3
SPSR	0x00000000

The memory location of input data.



```
0x00001000: DC 00 49 00 00 00 00 00
0x00001017: 00 00 00 00 00 00 00 00
```

The input start is at 0x00001000. Number of 1's are at R3 and 0's are at R4.

Result:

The experiments on compare operation has been performed and verified to be correct.

Experiment 7

Aim:

To write an ARM Assembly Language to multiply two numbers using repeated addition.

Tool Used:

Keil uVision4

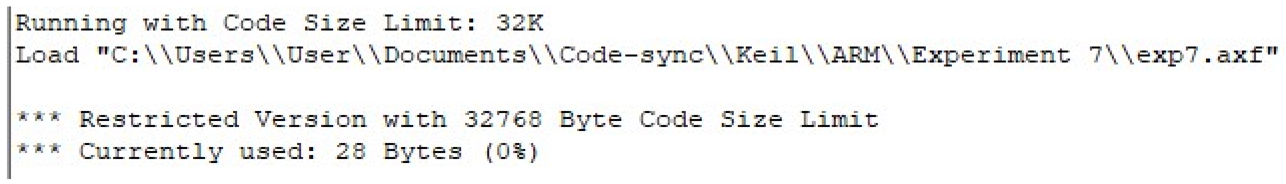
Theory:

One number can be used as counter and the other number can be decremented every loop. On every loop the 1st number is adder on to the result.

Code:

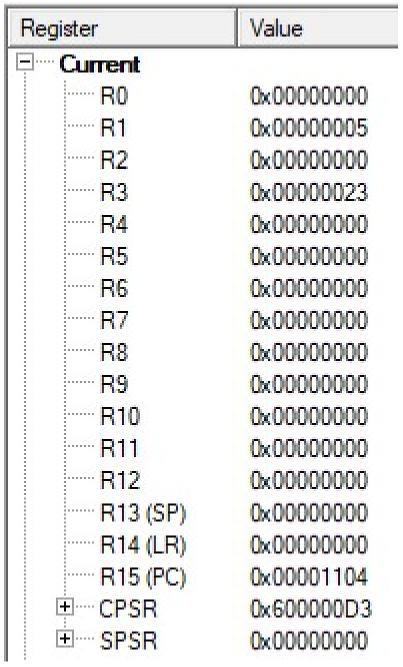
```
 AREA PROGRAM, CODE, READONLY
 ENTRY
MAIN
        LDR R0, =0X00001000 // location of input data
        LDR R1, [R0], #4 // loading 1st data
        LDR R2, [R0], #4 // loaing 2nd data
LOOP    ADD R3,R3,R1 // add 1st number to result
        SUBS R2,R2,#1 // decrementing number 2
        BNE LOOP //loop branch
        STR R3, [R0] // store the result in the memory location
        END
```

Output:



```
Running with Code Size Limit: 32K
Load "C:\\Users\\User\\Documents\\Code-sync\\Keil\\ARM\\Experiment 7\\exp7.axf"

*** Restricted Version with 32768 Byte Code Size Limit
*** Currently used: 28 Bytes (0%)
```

Register Contents



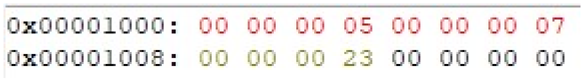
Register	Value
Current	
R0	0x00000000
R1	0x00000005
R2	0x00000000
R3	0x00000023
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00001104
CPSR	0x600000D3
SPSR	0x00000000

The memory location of

Input 1 = 0x00001000

Input 2 = 0x00001004

Output = 0x00001008



```
0x00001000: 00 00 00 05 00 00 00 07
0x00001008: 00 00 00 23 00 00 00 00
```

Result:

The experiments on multiplication operation has been performed and verified to be correct.