I. OBJECTIVE:

1. Perform Addition, Subtraction, Multiplication, and Division of two 16-bit numbers using immediate addressing mode and store the results using direct addressing mode.
2. Perform the following operations on two 8-bit data (**data1**, **data2**) given in memory locations and store the result in another memory location using indirect addressing mode.
    i. Swapping of nibble of **data1**
    ii. *Y*= (**data1** and **data2**) or (**data1** xor **data2**)
3. Find the Gray code of an 8-bit binary number.
4. Find the 2's complement of an 8-bit number.

II. PRE-LAB

- Explain the addressing modes involved in instructions.

For each objective in prelab describe the following points:

- Write the assembly code with a description (ex. Mov ax,3000h – ax<-3000h)
- Examine and analyze the input/output of assembly code.

PRE LAB

Objective 1: Perform Addition, Subtraction, Multiplication & division using two 16-bit number using immediate addressing and store the result in directly addressing

```
MOV AX, 4526H ; Immediate addressing mode, [AX] ← 4526H
MOV CX, AX ; [CX] ← [AX], [CX] ← 4526H
MOV BX, 220AH [BX] ← 220AH
ADD AX, BX; [AX] ← [AX] + [BX]
MOV [4500H], AX
MOV AX, CX ; [AX] ← [CX]
SUB AX, BX ; [AX] ← [AX] - [BX]
MOV [4502H] AX ; [4502H] ← [AX]
MOV AX, CX; [AX] ← [CX]
MUL BX
MOV [4504H], AX ; [4504H] ← AX
MOV [4506H], DX ; [4506H] ← DX
MOV AX, CX ; [AX] ← [CX]
MOV DX, 0000H; [DX] ← 0000H
DIV BX
MOV [4508H], AX; [4508H] ← [AX]
MOV [450AH], DX; [450AH] ← [DX]
HLT
```

Input

Input	Data
Ax, Cx	4526H
Bx	220AH
Dx	0000H

Output

Memory Location	Data
4500	30
4501	67
4502	1C
4503	23
4504	7C
4505	BF
4506	31
4507	09
4508	02
4509	00
450A	12
450B	01

Objective 2: Perform the following operations on two 8-bit data
(i) Swapping of nibble of data 1
(ii) Y = (data1 and data2) or (data1 x data2)

```
MOV SI, 5000H
MOV AL, [SI]
MOV CL, AL
MOV SI
MOV BL, [SI]
XOR AL, 044
INC SI
MOV [SI], AL
MOV AL, CL
AND AL, BL
XOR CL, BL
OR AL, CL
INC SI
MOV [SI], AL
HLT
```

Input

Memory location	Data
5000	AB
5001	DF

Output

memory location	Data
5002	BA
5003	AF

Objective 3: Gray code of an 8-bit binary number

```
MOV AL, [50004]
SHR AL, 01H
XOR AL, [50004]
MOV [5005H], AL
HLT
```

Input

Memory Location	Data
5000	54

Output

Memory location	Data
5000	7E

Objective 1: Find the 2's complement of an 8-bit number

```
MOV AL,[5000H]
NOT, AL
ADD AL, 01H
MOV [5001H], AL
HLT
```

Input

memory location	Data
5000	27

Output

memory location	Data
5001	D9

Different addressing modes involved in instructions are

(i) Immediate AM: The source operand is directly present in the instructio

(ii) Direct AM: The offset address is directly given in the instruction

(iii) Indirect AM: The offset address is given through a register

OBSERVATION

Objective 1:

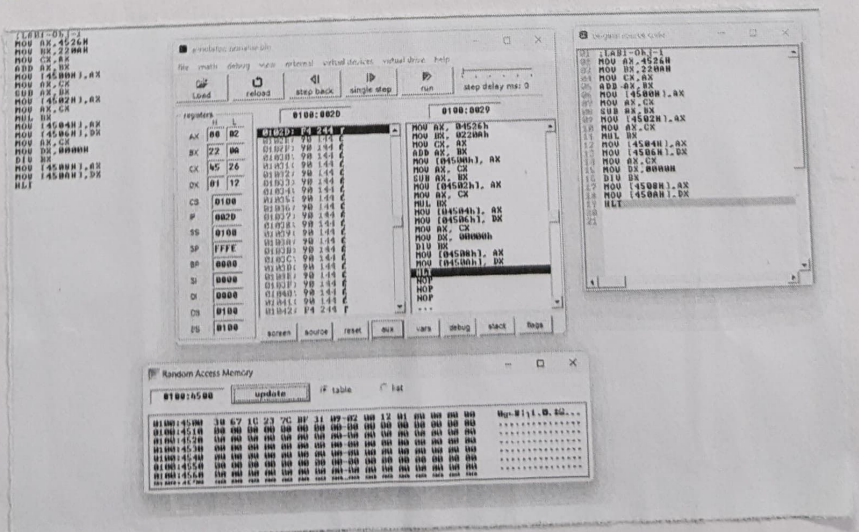


fig-1: Execution result of Sum, difference & Product of two e

Operand 1: 45284 Operand 2: 220AH	SUM 6730	DIFFERENCE 231C	Product	Q
	ML: 4500H: 30	ML: 4502: 1C	ML: 4504H; 7C	ML: 45

I. OBJECTIVE:

1. Find the sum and average of N 16-bit numbers.
2. Count no. of 0's in an 8-bit number.
3. Move a block of 16-bit data from one location to other.
4. Multiplication of two 16-bit numbers without using MUL instruction in direct addressing mode.

II. PRE-LAB

Note: For each objective in prelab describe the following points:

- Write the pseudocode.
- Write the assembly code with description (ex. Mov ax,3000h – ax<-3000h)
- Examine & analyze the input/output of assembly code.

III. LAB

Note: For each objective do the following job and assessment:

- Screen shots of the Assembly language program (ALP)
- Observations (with screen shots)

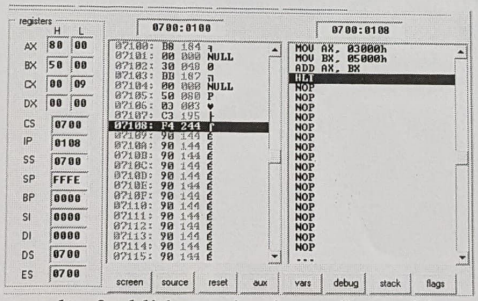


Fig. 1. Execution result of addition using immediate addressing mode of 8086 emulator.

From this result I have observed.....

Input:

Sl. No.	Memory Location	Operand (Data)
1		
2		
...		

Output:

Sl. No.	Memory Location	Operand (Data)
1		
2		
...		

PRE LAB :

Objective 1 : Find the sum and average of N 16-bit numbers

Pseudocode

→ First take the data (the numbers) as input you want to take

→ Then add the numbers one by one the answers is store in AX (Accumulator)

→ Then divide the sum by the no. of enputs. You get the average.

Assembly Code

```
MOV SI, 5000H
MOV CL, [SI]
MOV CH, 00H
MOV BX, CX
MOV AX, 000DH
MOV DX, 0000H
L2: INC SI
    INC SI
    ADD AX, [SI]
    JNC L1
    INC CH
L1: DEC CL
    JNZ L2
    INC SI
    INC SI
    MOV [SI], AX
    INC SI
    Mov [SI], CH
    Mov DL, CH
    DIV, BX
    INC SI
    INC SI
    MOV [SI] AX
    INC SI
    INC SI
    MOV [SI], DX
HLT
```

Analysis

(i) We take input through memory before execute the code.

(ii) Transfer it from memory to BX and one by one and it to AX

(iii) At last we get Sum of numbers AX

Objective 2: Count no. of 0's in an 8-bit number

Pseudocode

(i) Right Shift to the 8-bit number 1 by 1 when check the carry flag.

(ii) If the carry flag is 1 then the bit is 1 so don't increase the count that you store in another register

(iii) When carry is not generated then increase the count

(iv) After 8 timed right shift (as the 8-bit numbers) we get our desired result.

```
MOV BX, 5000H
MOV AL, [BX]
MOV CL, 00H
MOV CH, 08H
L2: SHR AL, 01H
    JC L1
    INC CL
L1: DEC CH
    JNZ L2
    INC BX
    MOV [BX], CL
    HLT
```

Objective 3: Move a block of 16-bit data from one location to another

Pseudocode

→ Take the data in the memory location that store SI
→ INC the SI, Also set the destination address in DX
→ MOV the data from M[SI] to M[Dx] through a general Purpose register

Assembly Code

```
MOV Ax, 2000H
Mov DS
MOV SI, 3000H
MOV CL,[SI]
INC SI
MOV DI , 5000H
L1 : MOV Bx ,[SI]
     Mov [DI] , Bx
     INC SI
     INC SI
     INC DI
     INC DI
     DEC CL
     JNZ L1
HLT
```

Analysis

(i) We store 3000H in SI, which is our Source address whose Onwards we gave data

(ii) We move the data at M(SI) to register then register to destinatio address

INPUT	Output
[5001H] → 12	M[3000H] ← Bx ← M[5001H]
[5002H] → 34	M[3001H] ← Bx ← M[5002H]
[5003H] → 56	M[3002H] ← Bx ← M[5003H]
[5004H] → 78	M[3003H] ← Bx ← M[5004H]

Objective 4: Multiplication of 16 bit numbers without using MUL Instruction

Pseudocode

→ Take 2 data as input on which you want to perform operation
→ Add the 1st data with ch shelf for "2nd data" times
→ The final result is the multiplication of that two numbers

Assembly Code

```
MOV BX [1000H]
MOV CX [1002H]
MOV DX, 0000H
MOV AX, 0000H
L1 : APD AX, BX
     JNC L2
     INC DX
L2 : DEC CX
     JNZ L1
     MOV [1004H], AX
     MOV [1006H], DX
HLT
```

Analysis

→ Bx Contain the Perst and Cx Contains the Second data
→ We add Ax with Bx for Cx times
→ then if in every addition Carry is generate the Carry bit store in Dx

So the result is

Bx ← [1000H] = 5634
Cx ← [1002H] = 000AH

56344 X 000AH = 035F08H

OBJECTIVE 1: Find the largest / smallest (8-bit numbers) from a given array of size N.

For largest Number

Pre-LAB

Assembly Code

```
.data
Array_length db 04h
 Value db 09h, 10h, 05h, 03h
 result db 00h
.Code
  MAIN PROC
    MOV Ax, data
    MOV DS, AX
    MOV CL, Array_length
    LEA SI, Value
    MOV AL, [SI]
 UP: DEC CL
    JZ next
    INC SI
    COMP AL, [SI]
    JNL UP
    MOV AL, [SI]
    JMP UP
next: LEA DI, result
     MOV [DI], AL
END MAIN
```

Smallest Number

Pre lab

Assembly Code

```
.data
array_length db 04H
array db 09H, 10H, 05H, 03H
result db 00H

.Code
MAIN PROC
    MOV Ax, data
    MOV DS, AX
    MOV CL, array_length
    LEA SI, array
    MOV AL, [SI]
UP: DEC CL
    JZ next
    INC SI
    CMP AL, [SI]
    JL UP
    MOV AL, [SI]
    JMP UP
next: LEA DI, result
      MOV [DI], AL
END MAIN
```

OBJECTIVE 2: Arrange the elements (8-bit numbers) of a given array of size N in ascending/descending order.

Ascending Order

Pre lab

Assembly Code

```
.data
  array_length db 04H
  array db 09H, 0CH, 06H, 01H

.Code
  MAIN PROC
    Mov AX, data
    Mov DS, AX
    Mov CL, array_length
    DEC CL
  UP2: MOV CH, CL
       LEA SI, array
  UP1: Mov AL, [SI]
       CMP AL, [SI+1]
       JC DOWN
       MOV DL, [SI+1]
       XCHG DL, [SI]
       Mov [SI+1], DL
  DOWN: INC SI
        DEC CH
        JNZ UP1
        DEC CL
        JNZ UP2

  END MAIN
```

Descending Order

Pre lab

Assembly Code

```
.data
 array-length db 04H
 array db 09H, 0CH, 06H, 01H

.code
 MAIN PROC
    MOV Ax, data
    MOV DS, Ax
    MOV CL, array-length
    DEC CL

UP2: MOV CH, CL
    LEA SI, array

UP1: MOV AL, [SI]
    CMP AL, [SI+1]
    JNC DOWN
    MOV DL, [SI+1]
    XCHG DL, [SI]
    MOV [SI+1], DL
DOWN: INC SI
    DEC CH
    JNZ UP1
    DEC CL
    JNZ UP2

END MAIN
```

I. OBJECTIVE:

1. Perform Addition and Subtraction of two 32-bit numbers using data processing addressing mode (with immediate data).
2. Perform Addition, Subtraction, and Multiplication of two 32-bit numbers using load/store addressing mode.
3. Perform the logical operations (AND, OR, XOR, and NOT) on two 32-bit numbers using load/store addressing mode.

II. PRE-LAB

Note: For each objective in prelab describe the following points:

- Write the pseudocode.
- Write the assembly code with a description (ex. Mov ax,3000h – ax<-3000h)
- Examine & analyze the input/output of assembly code.

III. LAB

Note: For each objective do the following job and assessment:

- Screenshots of the Assembly language program (ALP)
- Observations (with screenshots)

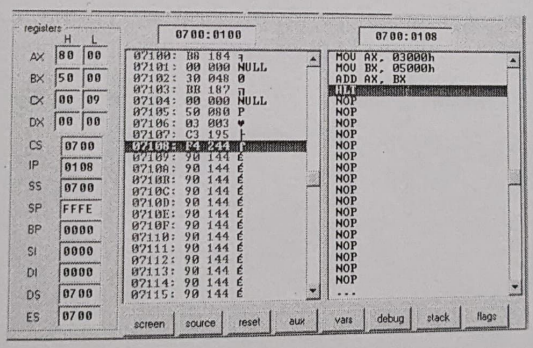


Fig. 1. Execution results of addition using immediate addressing mode of 8086 emulator.

From this result, I have observed……

Input:

Sl. No.	Memory Location	Operand (Data)
1		
2		
…		

Output:

Sl. No.	Memory Location	Operand (Data)
1		
2		
…		

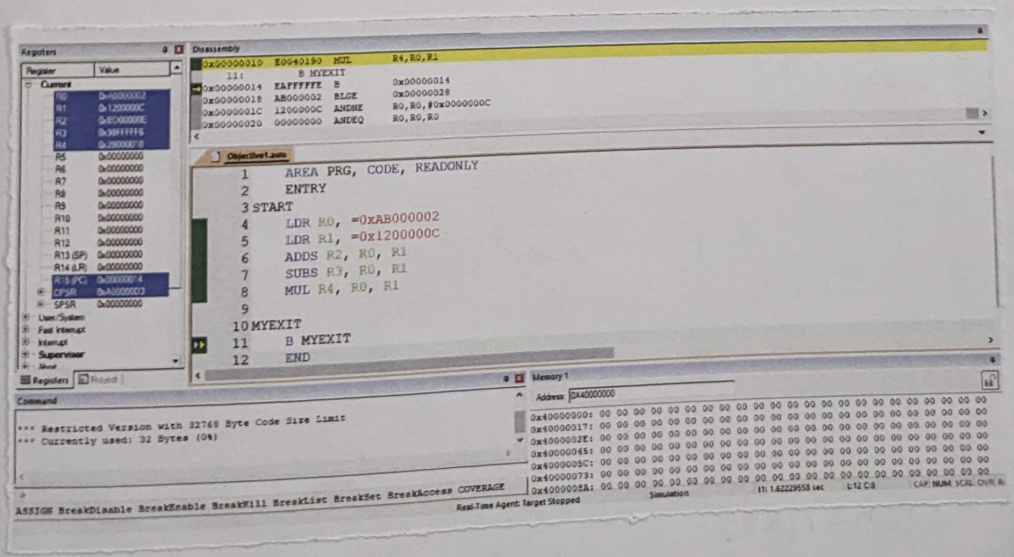
Objective-1 Perform Addition & Subtraction of two 32-bit numbers using data processing addressing mode (with immediate data)

Pre-Lab

Objective-1

```
        AREA PRG, CODE, READONLY
        ENTRY
START
        LDR R0, =0xAB000002
        LDR R1, =0x1200000C
        ADDS R2, R0, R1
        SUBS R3, R0, R1
        MUL R4, R0, R1
MYEXIT
        B MYEXIT
        END.
```

Input/output of ARM Assembly code



Obs. Observation Table

Input

Memory location	operand(data)
0x40000000	R0 ← 0xAB000002
0x40000004	R1 ← 0x1200000C

Output

Memory location	Operand (data)
0x40000008	R2 ← 0xBD00000E (ADD)
0x4000000C	R3 ← 0x98FFFFF6 (SUB)
0x40000010	R4 ← 0x28000018 (MUL)

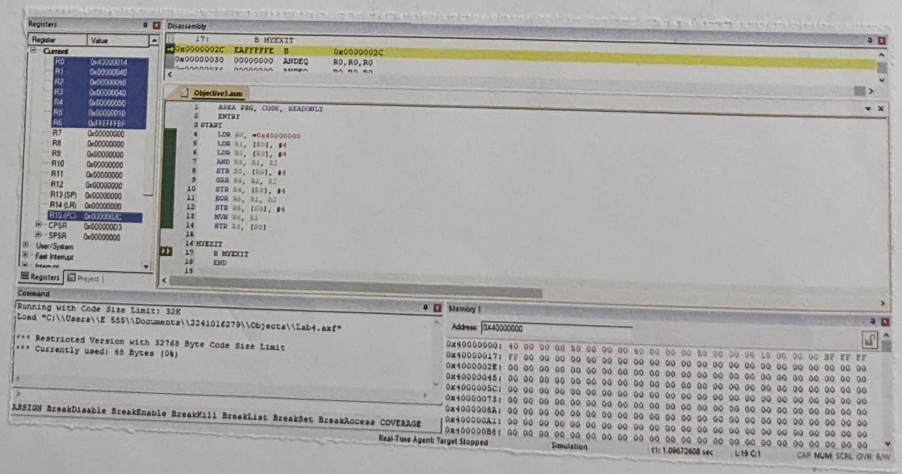
Objective 2: Perform addition, subtraction & multiplication of two 32-bit numbers using load/store addressing mode.

```
        AREA PRG, CODE, READONLY
        ENTRY
START
        LDR R0, =0x40000000
        LDR R1, [R0], #4
        LDR R2, [R0], #4
        ADDS R3, R1, R2
        STR R3, [R0], #4
        SUBS R4, R1, R2
        STR R4, [R0], #4
        MUL R5, R1, R2
        STR R5, [R0]
MYEXIT
        B MYEXIT
        END
```

Objective 3: Perform the logical operations (AND, OR, XOR, & NOT) on two 32-bit numbers using load / store addressing mode.

```
        AREA PRG, CODE, READONLY
        ENTRY
START
        LDR R0, =0x40000000
        LDR R1, [R0], #4
        LDR R2, [R0], #4
        AND R3, R1, R2
        STR R3, [R0], #4
        ORR R4, R1, R2
        STR R4, [R0], #4
        EOR R5, R1, R2
        STR R5, [R0], #4
        MVN R6, R1
        STR R6, [R0]
MYEXIT
        B MYEXIT
        END
```

Input / output of ARM Assembly Code



Objective 1: Find the largest number from a given array of size N using ARM assembly language

Assembly Code

```
.global _Start
_Start:
    @ Largest number from a given array
    ldr r0, =Count
    ldr r1, [r0]
    mov r4, #0x00
    ldr r2, =array
back: ldr r3, [r2], #4
    cmp r4, r3
    bgt fwd
    mov r4, r3
fwd: subs r1, r1, #01
    bne back
    str r4, [r2]
exit: b exit

.data
Count: .word 0x05
array: .word 0x15, 0x85, 0x45, 0x10, 0x4F
Result:
```

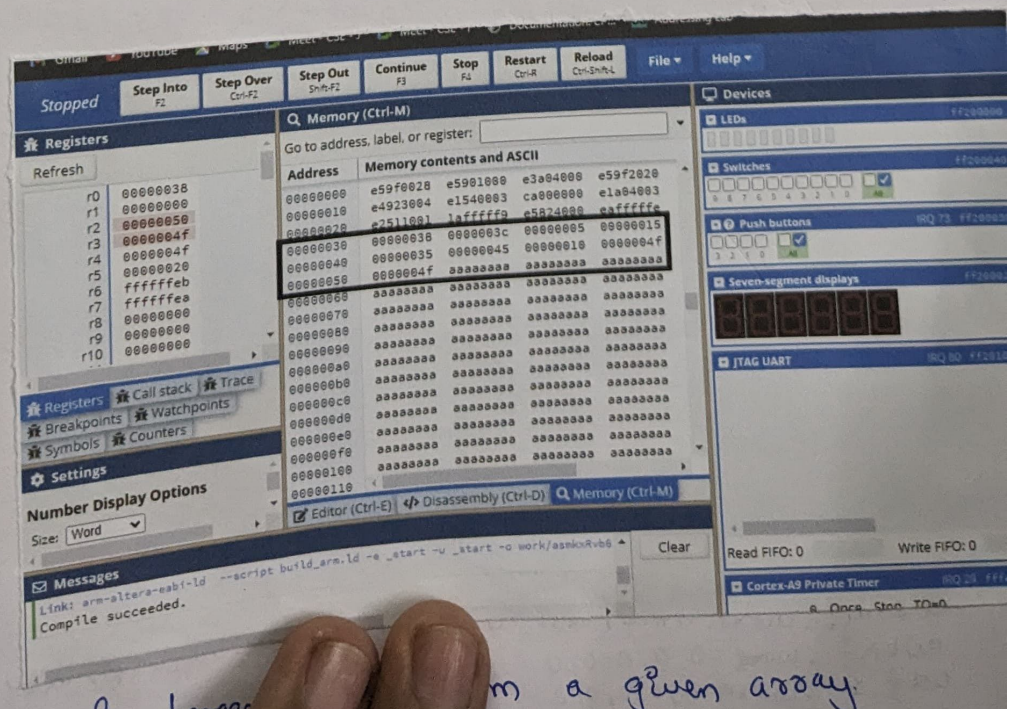


fig: larger ... m a given array.

Observation Table

Input

Memory Location	Data
00000038	00000005
r4	00000000
0000003C	00000015
00000040	00000035
00000044	00000045
00000048	00000010
0000004C	0000004f

Output

Memory Location	Data
00000050	00000040

Objective 2: Separate Even numbers and Odd numbers in a given array of size

```
.global _Start
_start:
        ldr r0, =Count
        ldr r1, [r0]
        ldr r3, =array
        ldr r4, =even
        ldr r5, =odd
back:   ldr r6, [r3], #4
        ands r7, r6, #1
        beq fwd
        Str r6, [r5], #4
        b fwd1
fwd:    Str r6, [r4], #4
fwd1:   Subs r1, r1, #01
        bne back
exit:   b exit
.data
count:  .word 0x07
array:  .word 0x15, 0x35, 0x32, 0x45, 0x10, 0x4f, 0x34
even:   .word 0,0,0,0,0
odd:    .word 0,0,0,0,0
```

Result:

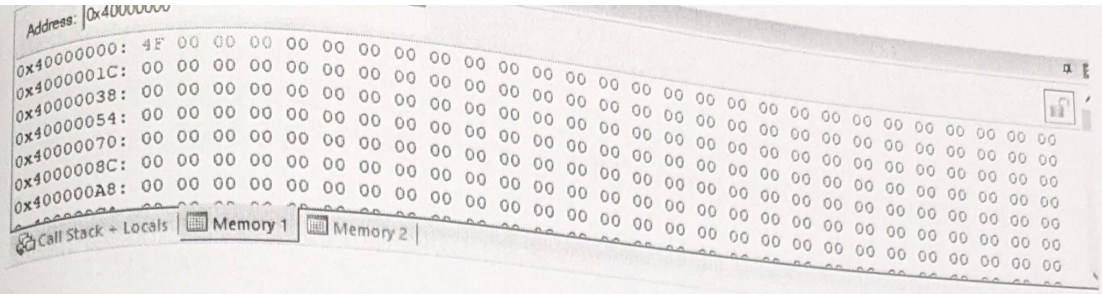
Lab-7:

Objective 1: **Find the Largest number from a given array of size N using ARM assembly Language.**

Program:

```
        AREA PROG1, CODE,READONLY
        ENTRY
START
        ;Largest number from a given array
        ldr r0,=count
        ldr r1,[r0]        ; r1= array size
        ldr r2,=array
   ldr r3, [r2],#4
back
        subs r1,r1,#01
        beq fwd
        ldr r4,[r2],#4
        cmp r3,r4
        bgt back
        mov r3,r4
        b back
fwd
    ldr r5,=RESULT
        str r3,[r5]
exit b exit

count DCD 0x07
array
                DCD 0x15 ; DCD= Define Constant Double-words(32-bit)
                DCD 0x35 ; DCD directive allocates one or more words of memory, aligned
                         ; on 4-byte boundaries
                DCD 0x32
                DCD 0x45
                DCD   0x10
                DCD 0x4f
                DCD 0x34
        AREA DATA2,DATA,READWRITE ; TO STORE RESULT IN GIVEN ADDRESS
RESULT
        DCD 0X0
        END
```



Objective-2 : Separate Even numbers and odds numbers in an array of size N using ARM Assembly language.

Program:

```
        AREA prog2, CODE, READONLY
        ENTRY      ;Mark first instruction to execute
START
        ldr r0,=count
        ldr r1,[r0]
        ldr r3,=array ; r3 = base address of array=array[0]
        ldr r4,=even  ; r4=base address of even data locations as constant = even[0]
                                                        ; = 0x40000000
        ldr r5,=odd   ; r5=base address of odd data locations as constant = odd[0]
                                                        ; = 0x4000001c
back
        ldr r6, [r3],#4
        ands r7,r6,#1
        beq fwd
        str r6,[r5],#4
        b fwd1
fwd
```

```
        str r6,[r4],#4
fwd1
        subs r1,r1,#01
        bne back

exit b exit
; Array declaration
count DCD 0x07
array
                DCD 0x15
                DCD 0x35
                DCD 0x32
                DCD 0x45
                DCD   0x10
                DCD 0x4f
                DCD 0x34
        AREA DATA2,DATA,READWRITE ; TO STORE RESULT IN GIVEN ADDRESS
even
        DCD 0X0
        DCD 0X0
        DCD 0X0
        DCD 0X0
        DCD 0X0
        DCD 0X0
        DCD 0X0
 odd
        DCD 0X0
        DCD 0X0
        DCD 0X0
        DCD 0X0
        DCD 0X0
        DCD 0X0
        DCD 0X0
        END
```

Result:

Input Location: