

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. $Y = (\text{data1} \text{ and } \text{data2}) \text{ or } (\text{data1 xor data2})$
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 numbers using Immediate addressing and store the result in directly addressing

MOV AX, 4526H ; Immediate addressing mode $[AX] \leftarrow 4526H$

MOV CX, AX ; $[CX] \leftarrow [AX]$, $[CX] \leftarrow 4526H$

MOV BX, 220AH $[BX] \leftarrow 220AH$

ADD AX, BX; $[AX] \leftarrow [AX] + [BX]$

MOV [4500H], AX

MOV AX, CX; $[AX] \leftarrow [CX]$

SUB AX, BX; $[AX] \leftarrow [AX] - [BX]$

MOV [4502H], AX; $[4502H] \leftarrow [AX]$

MOV AX, CX; $[AX] \leftarrow [CX]$

MUL BX

MOV [4504H], AX; $[4504H] \leftarrow AX$

MOV [4506H], DX; $[4506H] \leftarrow DX$

MOV AX, CX; $[AX] \leftarrow [CX]$

MOV DX, 0000H; $[DX] \leftarrow 0000H$

DIV BX

MOV [4508H], AX; $[4508H] \leftarrow [AX]$

MOV [450AH], DX; $[450AH] \leftarrow [DX]$

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) $Y = (\text{data1} \text{ and } \text{data2}) \text{ or } (\text{data1} \times \text{data2})$

MOV SI, 5000H

MOV AL, [SI]

MOV CL, AL

MOV SI

MOV BL, [SI]

XOR AL, 04H

INC SI

MOV [SI], AL

MOV AL, CL

AND AL, BL

XOR CL, BL

OR AL, CL

INC SI

MOV [SI], AL

HLT

<u>Input</u>	
Memory Location	Data
5000	AB
5001	DF

<u>Output</u>	
Memory Location	Data
5002	BA
5003	AF

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

MOV AL, [5000H]

SUB AL, 01H

XOR AL, [5000H]

MOV [5005H], AL

HLT

<u>Input</u>	
Memory Location	Data
5000	54

<u>Output</u>	
Memory Location	Data
5000	7E

Objective 4: Find the 2's complement of an 8-bit number

```

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

```

<u>Input</u>	<u>Output</u>		
<u>memory location</u>	<u>Data</u>	<u>memory location</u>	<u>Data</u>
5000	27	5001	D9

Different addressing modes involved in instructions are

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

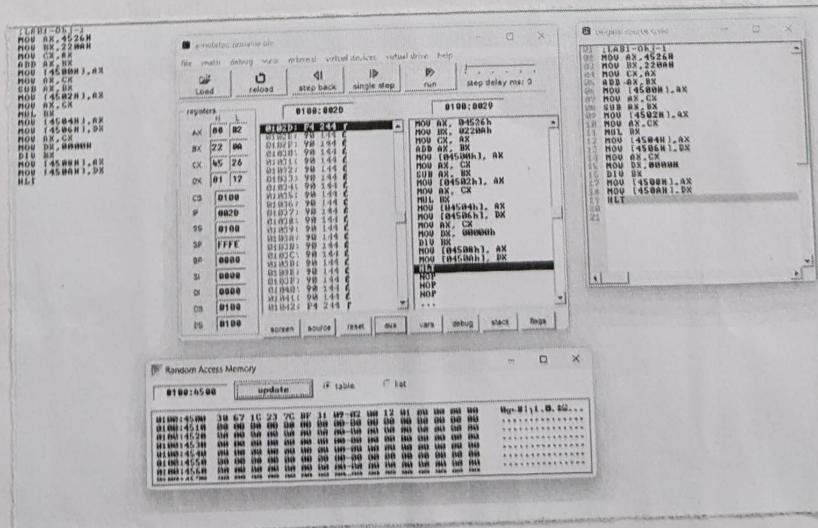


fig. 1: Execution result of Sum, difference & Product of two 8-bit numbers

OPCODE 1: 4528H	SUM	DIFFERENCE	Product	Q
OPCODE 2: 020AH	6730	231C	ML: 4504H; 7C	ML: 4511H

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)

The screenshot shows the 8086 emulator interface. On the left, a table of registers is displayed with their H and L values. The registers are AX, BX, CX, DX, CS, IP, SS, SP, BP, SI, DI, and DS, all set to 0700. The right side shows two windows of assembly code. The top window is labeled 0700:0100 and the bottom window is labeled 0700:0108. The assembly code consists of several NOP instructions (opcode F4) and some MOV and ADD instructions. The CPU status register (CS) is highlighted in the registers table.

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 either from a file or from user.
- Then add the numbers one by one the answer is stored in AX (Accumulator)
- Then divide the sum by the no. of inputs. 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 A

(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 number) we get our desired result.

```
MOV BX, 5000H  
MOV AL, [BX]  
MOV CL, 00H  
MOV CH, 08H  
L2: SHR AL, 01H  
    JC L1 L1  
    INC CL  
L1: DEC CL  
    JNZ L2  
    JNC 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 where onwards we gave data

(ii) We move the data at $M(SI)$ to registers then registers to destination address

INPUT

$[5001H] \rightarrow 12$
 $[5002H] \rightarrow 34$
 $[5003H] \rightarrow 56$
 $[5004H] \rightarrow 78$

Output

$M[3000H] \leftarrow BX \leftarrow M[5001H]$
 $M[3001H] \leftarrow BX \leftarrow M[5002H]$
 $M[3002H] \leftarrow BX \leftarrow M[5003H]$
 $M[3003H] \leftarrow BX \leftarrow M[5004H]$

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

Pseudocode

- Take 2nd 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: ADD AX, BX
    JNC L2
    INC DX
L2: DEC CX
    JNZ L1
    MOV [1004H], AX
    MOV [1006H], DX
HLT
```

Analysis

- BX Contain the first 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 stored in DX

So the result is

$$\begin{aligned} BX &\leftarrow [1000H] = 5634 \\ CX &\leftarrow [1002H] = 0001010 \end{aligned}$$

$$5634H \times 0001010 = 035F08H$$

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

For largest Number

Poe-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]

JNLL UP

MOV AL, [SI]

JMP UP

Next: LEA DI, result

MOV [DI], AL

END MAIN

(addr)	Value	Format
0000:0130	0000000000000000	0000000000000000
0000:0140	0000000000000000	0000000000000000
0000:0150	0000000000000000	0000000000000000
0000:0160	0000000000000000	0000000000000000
0000:0170	0000000000000000	0000000000000000
0000:0180	0000000000000000	0000000000000000
0000:0190	0000000000000000	0000000000000000



Smallest Number

For lab

Assembly Code

.data

array-length db 04H

array db 09H, 10H, 05H, 03H

result db 00H

.Code

MAIN PROC

MOV AX, data

MOU DS, AX

MOV CL, array-length

LEA SI, array

MOV AL, [SI]

UP: DEC CL

JZ Next

JNC 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 orders.

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

Pse 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

UPI: 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 UPI

DEC CL

JNZ UP2

END MAIN

OBJECTIVE:

I.

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)

The screenshot shows the execution results of an assembly program. The assembly code is:

```
MOV AX, 03000h  
MOV BX, 05000h  
ADD AX, BX  
HLT
```

The registers are set as follows:

Registers	H	L
AX	80	00
BX	50	00
CX	00	09
DX	00	00
CS	0700	07100: F4 244 E
IP	0108	07100: F4 244 E
SS	0700	07100: 90 144 E
SP	FFFE	07100: 90 144 E
BP	0000	07100: 90 144 E
SI	0000	07100: 90 144 E
DI	0000	07100: 90 144 E
DS	0700	07100: 90 144 E
ES	0700	...

Below the assembly code window are several buttons: screen, source, reset, aux, vars, debug, stack, and flags.

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

The screenshot shows a debugger interface with three main tabs: Registers, Disassembly, and Memory.

Registers Tab: Displays the current state of various ARM registers. The CPSR register is highlighted.

Disassembly Tab: Shows the assembly code listing:

```
1 AREA PRG, CODE, READONLY
2 ENTRY
3 START
4 LDR R0, =0xAB000002
5 LDR R1, =0x1200000C
6 ADDS R2, R0, R1
7 SUBS R3, R0, R1
8 MUL R4, R0, R1
9
10 MYEXIT
11 B MYEXIT
12 END
```

Memory Tab: Displays memory starting at address 0x40000000. The memory dump shows the values of the registers R0 through R4.

Observation Table

<u>Input</u>	
<u>Memory location</u>	<u>operand(data)</u>
0x40000000	R0 \leftarrow 0xAB000002
0x40000004	R1 \leftarrow 0x1200000C

<u>Output</u>	
<u>Memory location</u>	<u>Operand (data)</u>
0x40000008	R2 \leftarrow 0xBD00000E (ADD)
0x4000000C	R3 \leftarrow 0x98FFFFFF6 (SUB)
0x40000010	R4 \leftarrow 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

The screenshot shows a debugger window with three main panes:

- Registers:** Shows CPU registers R0-R7, R11, R12, R14, CPSR, and SP, all initialized to 0x00000000.
- Memory:** A dump of memory starting at address 0x40000000, showing the assembly code and its binary representation.
- Command:** Displays the assembly code listing with line numbers 1 through 19.

```

1 AREA PRG, CODE, READONLY
2 ENTRY
3 START
4 LDR R0, =#0x40000000
5 LDR R1, [R0], #4
6 LDR R2, [R0], #4
7 AND R3, R1, R2
8 STR R3, [R0], #4
9 ORR R4, R1, R2
10 STR R4, [R0], #4
11 EOR R5, R1, R2
12 STR R5, [R0], #4
13 MVN R6, R1
14 STR R6, [R0]
15
16 MYEXIT
17 B MYEXIT
18 END
19

```

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

Assembly Code

.global _Start

_Start:

@largest numbers 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:

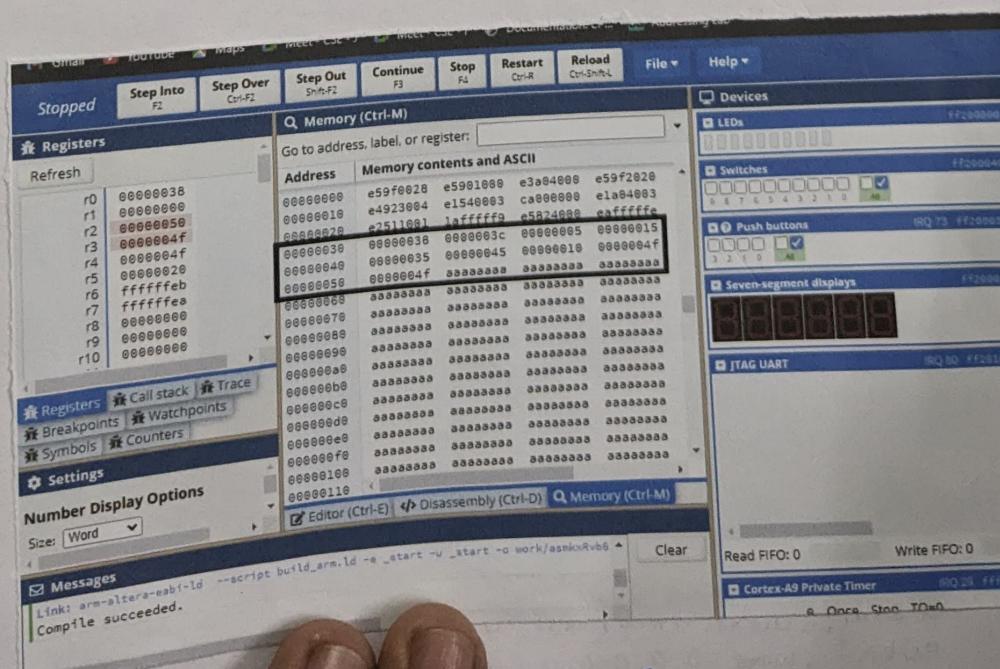


fig: larger

m a given array

Observation Table

<u>Input</u>		<u>Output</u>	
Memory Location	Data	Memory Location	Data
000000088	00000005	0000 0050	0000004f
r4	00000000		
00000003C	00000015		
000000040	00000035		
000000044	00000045		
000000048	00000010		
00000004C	0000004f		

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

and 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

odata.

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:

Address:

Lab-7:
Objective 1: Find the Largest number from a given array of size N using ARM assembly Language.

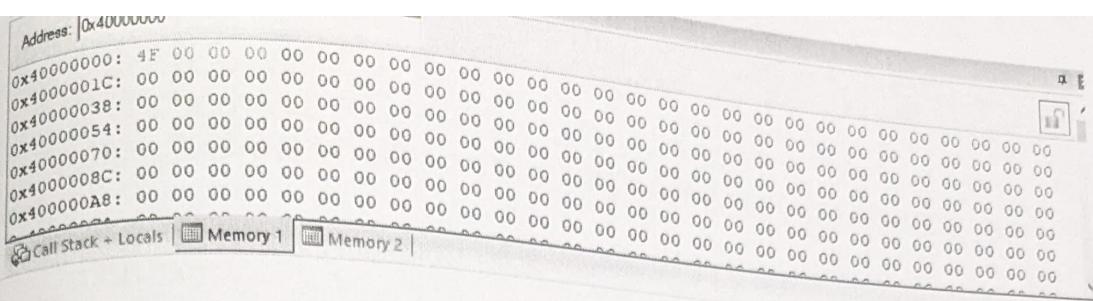
program:

```
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

odd
    DCD 0X0
    DCD 0X0

END
```

Result:

Input Location: