EXPERIMENT 3

OBJECTIVE: Program involves counting even and odd numbers from a given array. The objective of this program is to give an overview of the string instructions of 8086.

Aim: a. To count even and odd numbers from an array of 10 numbers.
b. To find average of 10 numbers

Theory: A string is a series of bytes stored sequentially in the memory. The SRC element is taken from the data segment using and SI register. The destination element is in extra segment pointed by DI register. These registers are incremented or decremented after each operation depending upon the direction flag in flag register.

Some of the instructions useful for program are,
1) **CLC**- the instruction clears the carry flag.
2) **RCR** destination, count- Right shifts the bits of destination. LSB is shifted into CF. CF goes to MSB. Bits Are shifted counts no of times.

3) **JC**: jump to specified location.
4) **INC/DEC** destination: add/subtract 1 from the specified destination.

5) **JMP**label: The control is shifted to an instruction to which label is attached.

6) **JNZ**label: The control is shifted to an instruction to which label is attached if ZF = 0.

Algorithm:
1. Initialize the data segment.
2. Initialize the array.
3. Load the effective address of an array in any index register.
4. Load total number of elements of the array in any register.
5. Initialize any two registers as counter for even and odd numbers to zero.
6. Load first element of an array in any general purpose register.
7. Shift/rotate the contents of loaded register to right.
8. If CF=1 increment counter for odd numbers otherwise increment counter of even numbers.
9. Store the value of even and odd counter register to two memory locations.
10. Stop.

Algorithm:
1. Initialize the data segment.
2. Initialize the array.
3. Load the effective address of an array in any index register.
4. Load total number of elements of the array in Cl register.
5. Load Al with zero
6. Add the element of array to the Al register and store sum in Al
7. Increment the index and decrement the number of elements of array in Cl register
8. Check whether Cl register has become Zero, if no repeat 6 else stop

Code:

a) Even Odd Count

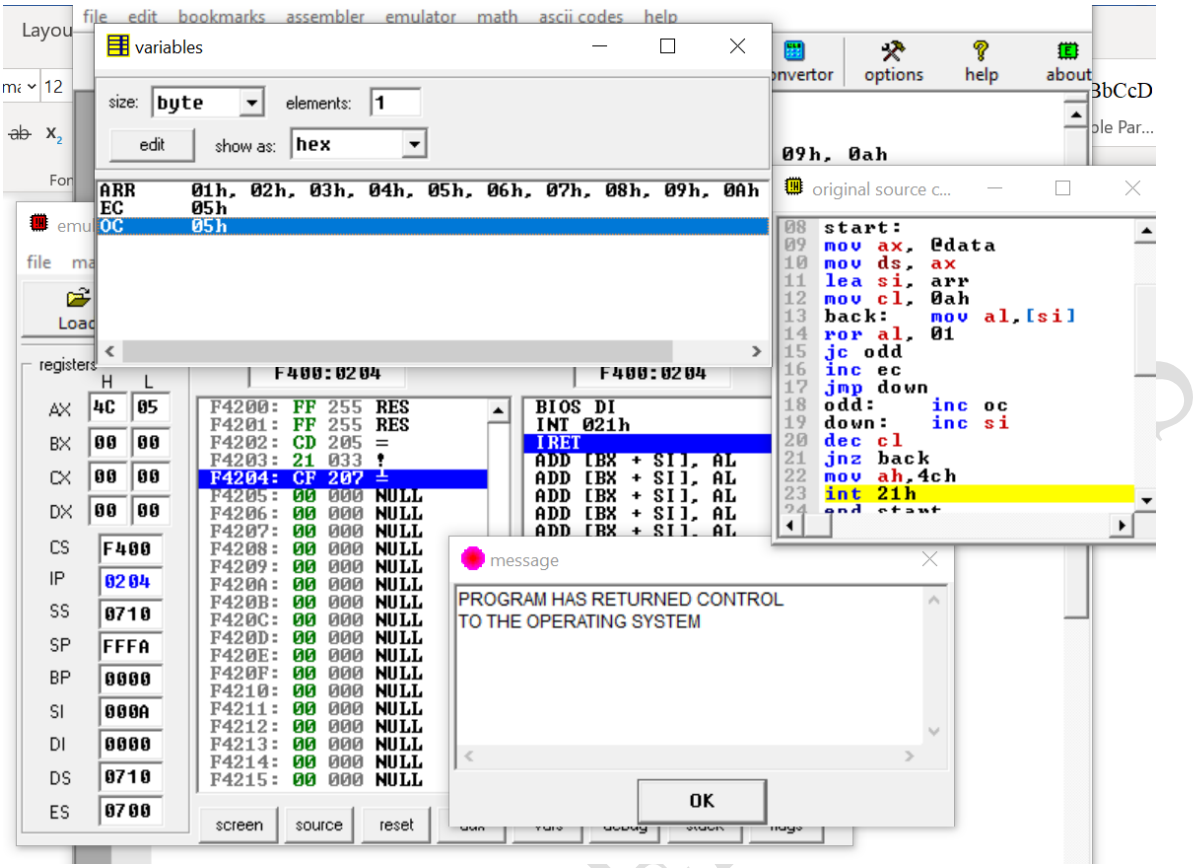
```
.8086
.model small
.data
arr db 01h, 02h, 03h, 04h, 05h, 06h, 07h, 08h, 09h, 0ah
ec db ?
oc db ?
.code
start:
mov ax, @data
mov ds, ax
lea si, arr
mov cl, 0ah
back:   mov al,[si]
        ror al, 01
        jc odd
        inc ec
        jmp down
odd:    inc oc
down:   inc si
        dec cl
        jnz back
mov ah,4ch
int 21h
   end start
```

b) Average of 10 numbers

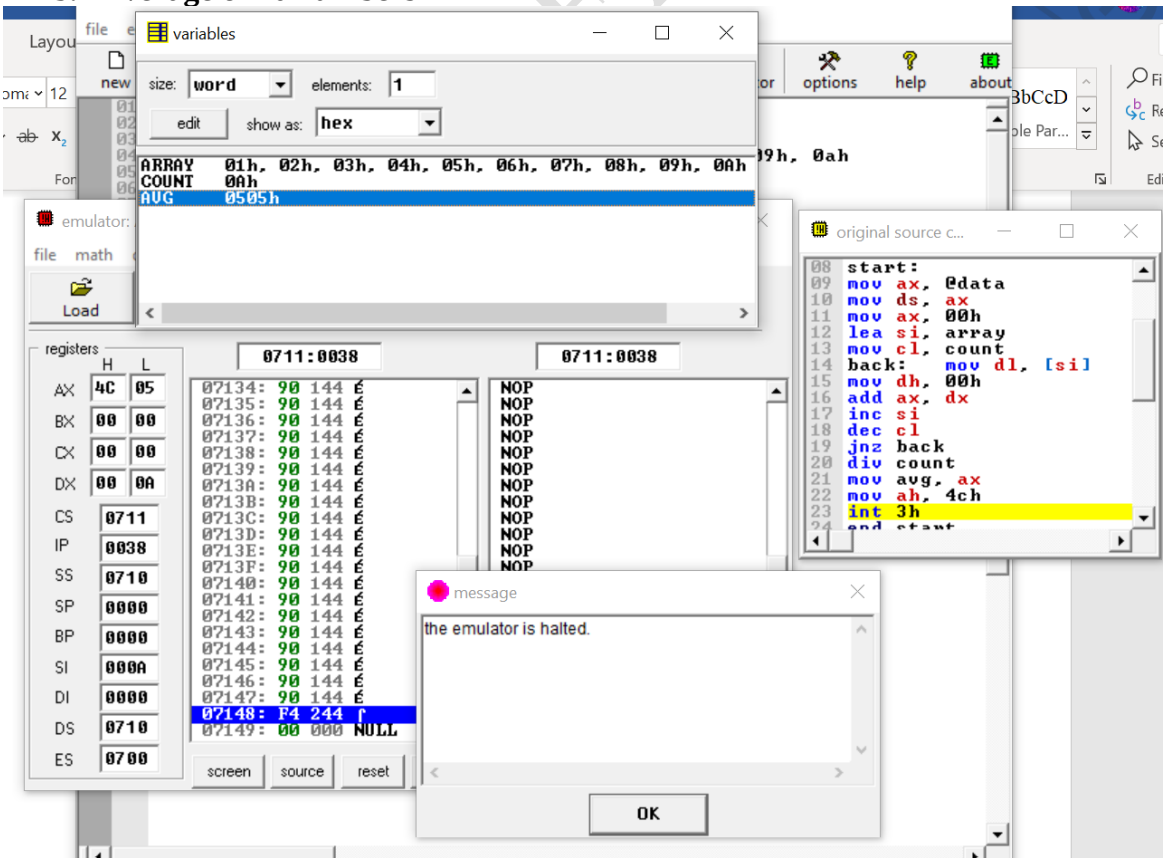
```
.8086
.model small
.data
array db 01h, 02h, 03h, 04h, 05h, 06h, 07h, 08h, 09h, 0ah
count db 0ah
avg dw ?
.code
start:
mov ax, @data
mov ds, ax
mov ax, 00h
lea si, array
mov cl, count
back:   mov dl, [si]
        mov dh, 00h
        add ax, dx
        inc si
        dec cl
        jnz back
div count
mov avg, ax
mov ah, 4ch
int 3h
end start
```

Screenshot of Output:

a. Even Odd Count



b. Average of 10 numbers



Post Lab:

1) Explain Processor control instructions.
   Ans.
   - **HLT** – Halt processing

     HLT instruction causes the 8086 to stop fetching and executing instructions. The 8086 will enter a halt state. The different ways to get the processor out of the halt state are with an interrupt signal on the INTR pin, an interrupt signal on the NMI pin, or a reset signal on the RESET input.

   - **NOP** – Perform No Operation

     The NOP instruction can be used to increase the delay of a delay loop. When hand coding, a NOP can also be used to hold a place in a program for an instruction that will be added later. NOP does not affect any flag.

   - **ESC** – Escape

     This instruction is used to pass instructions to a coprocessor, such as the 8087 Math coprocessor, which shares the address and data bus with 8086. Instructions for the coprocessor are represented by a 6-bit code embedded in the ESC instruction.

2) Describe the difference between shift and rotate instruction with appropriate example.
   Ans.
   The difference between Shift and Rotate instructions is that the rotate cycles the bits around going out one side and coming in the other, while shift rotates the bits out one side or the other leaving the space where the rotated bits where either unchanged or zeroed.
   Below are few examples of the same

SHL/SAL Left shifts the bits of destination. MSB shifted into carry. LSB gets 0 Eg. SAL reg1, count	**ROL** Left shifts the bits of destination. MSB shifted into the carry. MSB also goes to LSB. Eg. ROL reg, count
SHR Right Shifts the bits of destination. MSB gets a 0. LSB shifted into carry. Eg. SHR reg, count	**ROR** Right shifts the bits of destination. LSB shifted carry. LSB also goes to MSB Eg. ROR reg, count
SAR Right shifts the bits of destination. MSB placed in MSB itself. LSB shifted into carry. Eg. SAR reg, count	**RCR** Right shift the bits of destination. LSB shifted into the CF. CF goes to MSB. Eg. RCR reg, count