Write an assembly language program to perform pairwise scan operation on an array such that:
If second element of the pair is even, then multiply 1st and 2nd element through bit manipulation and store the result in place of the first element. If second element of the pair is odd, then add 1st and 2nd element and store the result in the location ofthe first element. If the array contains odd number of elements, then save the last element as it is.

```
[org 0x0100]
jmp start
data: dw 1, 8, 4, 2, 3, -1
start: mov bx,0
loop1: cmp word [data+bx], -1 ;check to stop loop
jz end
cmp word [data+bx+2], -1
jz end
mov ax, [data+bx+2] ; check next element either even or odd
shr ax, 1
jc odd
even: mov cl, 8 ; multiply by using bit operations
mov ax, [data+bx]
mov dx, [data+bx+2]
mov word [data+bx], 0 ; set result on first place to zero
chkbit: shr dx, 1
jnc skip
add [data+bx], ax
skip: shl ax, 1
dec cl
jnz chkbit
jmp next
odd: mov ax, [data+bx+2] ; add elements in case of odd
add [data+bx], ax
next: add bx, 4 ; iterate to next elements
jmp loop1
end: mov ax, 0x4c00
int 0x21
```

Second version of Pairwise Scan:

if second element of the pair is even, then divide 2nd element by 4 through bit manipulation and store the result in place of the second element. If second element of the pair is odd, multiply 2nd element by 2 and store the result in the location of the second element. If the array contains odd number of elements, then save the last element as it is.

```
[org 0x0100]
jmp start
```

```
data: dw 1, 8, 4, 2, 3, -1
start: mov bx,0
loop1: cmp word [data+bx], -1 ;check to stop loop
jz end
Mov ax,[data+bx]
Shr ax,1
Jc odd
Even:
Shr word[data+bx+2],2
Jmp next
Odd:
Shl word[data+bx+2],1
next: add bx, 4 ; iterate to next elements
jmp loop1
end: mov ax, 0x4c00
int 0x21
```

Write an assembly program, such that given an array of ten integers (each integer is stored as word), your program finds and stores the sum of unique elements of array in a memory label called "sum" (defined word)

```
[org 0x0100]
jmp start
data: dw 3,3,3,3,3  ; Array of numbers to be sorted
swap: db 0          ; Swap flag
sum: dw 0           ; Sum variable
start:
mov bx, 0           ; Initialize array index to zero
mov byte [swap], 0  ; Reset swap flag to no swaps
 Bubble sort loop
loop1:
mov ax, [data+bx]     ; Load the number at data[bx] into ax
cmp ax, [data+bx+2]   ; Compare with the next number at data[bx+2]
jbe noswap            ; Jump to noswap if no swap is needed
mov dx, [data+bx+2]   ; Load the second element into dx
mov [data+bx+2], ax   ; Swap the first and second numbers
mov [data+bx], dx     ; Store the second number in the first position
mov byte [swap], 1    ; Flag that a swap has been done
noswap:
add bx, 2             ; Advance bx to the next index
cmp bx, 8             ; Check if we are at the last index
jne loop1             ; If not, compare the next two elements
cmp byte [swap], 1    ; Check if a swap has been done in this pass
je start              ; If a swap occurred, make another pass
 Sum calculation loop
mov bx, 0             ; Reset array index to zero
```

```
mov cx, [data+bx]     ; Store the first element in cx (sum variable)
l1:
mov ax, [data+bx]     ; Store the current element in ax
cmp ax, [data+bx+2]   ; Compare with the next element
je skip               ; If equal, skip adding (to avoid duplicates)
add cx, [data+bx+2]   ; Add the current element to the sum
skip:
add bx, 2             ; Advance bx to the next index
cmp bx, 8             ; Check if we are at the last index
jne l1                ; If not, continue summing
mov [sum], cx         ; Store the final sum in the sum variable
mov ax, 0x4c00        ; Terminate the program
int 21h
```

Write an assembly language program that processes a Given_Array (of bytes), calculates pairwise sum of its elements and saves the result in PairwiseSumArray. First element is paired with the last element, the 2nd element is paired with the 2nd last element and so on. See the Sample Run below for detail.

```
[org 0x100]
mov cx, ArrSize ; Load the size of the array into CX
mov si, 0       ; Initialize source index to 0
mov di, cx-1    ; Initialize destination index to ArrSize-1 (last element)
mov bx, Given_Array ; Load the address of Given_Array into BX
mov dx, PairwiseSumArray ; Load the address of PairwiseSumArray into DX
start:
          mov al, [bx+si] ; Load the current element from Given_Array into AL
          add al, [bx+di] ; Add the corresponding element from the other end of the array
          mov [dx+si], al ; Store the result in PairwiseSumArray
      inc si          ; Increment source index
          dec di          ; Decrement destination index
    cmp si, di      ; Have we reached the middle of the array?
          jae done        ; If yes, we're done
   jmp start       ; Otherwise, continue with the next pair
done:
   mov ax, 0x4c00
          int 0x21

ArrSize db 8 ; Define the size of the array
Given_Array db 10, 2, 3, 4, 50, 62, 70, 8 ; Define the Given_Array
PairwiseSumArray db 8 dup(0) ; Define the PairwiseSumArray with space for 8 elements
```

Q) Write a program to swap every pair of bits in the AX register i.e. swap bit # 0 with bit # 1, bit # 2 with bit # 3 and so on.

```
mov ax, 0b1011001001011101
mov cx, 8
mov bx, ax

l1:
      rcr di, 1
      rcr bx, 2
      rcr ax, 2
      jnc l2
      mov di, 1
      jmp l3
l2:
      mov di, 0

l3:
      sub cx, 1
      jnz l1
mov ax, 0x4c00
int 0x21
```

Q)**AX contains a non-zero number. Count the number of ones in it and store the result back in AX. Repeat the process on the result (AX) until AX contains one. Calculate in BX the number of iterations it took to make AX one. For example BX should contain 2 in the following case: AX = 1100 0101 1010 0011 (input – 8 ones) AX = 0000 0000 0000 1000 (after first iteration – 1 one) AX = 0000 0000 0000 0001 (after second iteration – 1 one) STOP**

```
[org 0x0100]
mov ax, 0xC5A3
mov dx, 0
mov bx, 0
mov si, 16
mov cx, 0
mov dx, ax
jmp start
loop1:
sub si, 1
jnz start
jz loop2
start:
shr dx, 1;
jnc loop1
add cx, 1
sub si, 1
jnz start;
loop2:
mov ax, 0
```

```
add ax, cx
add bx, 1
mov cx, 0
add si, 16
mov dx, ax
cmp ax, 1
je exit
jne loop1;
exit:
mov ax, 0x4C00
int 0x21
```

Q) **PALINDROME :**

[org 0x0100]

```
mov si , 0
mov di , [arrsize]
add di , di
sub di , 2

Check:
        mov ax , [arr + si]
        cmp ax , [arr + di]
        jne No
      add si , 2
        sub di , 2
        cmp si , di
        jae Yes
        jmp Check
No:     mov ax , -1
        jmp exit
Yes:    mov ax , 1
exit:   mov ax , 0x4C00
        int 0x21
arr : dw 1 , 2 , 7 , 4 , 5 , 4 , 3 , 2 , 1
arrsize : dw 9
```

Q) Binary Search, if found set AX=1 otherwise AX=0

```
[org 0x0100]
Jmp start1
Data: db 1,2,3,4,5,6,7,8,9,10,11
Start: db 0
End: db 10
Key: db -1
Start1: mov al,[key]
Loop1: mov cl,[start]
Cmp cl,[end]
```

```
Ja end1
end1:
Mov dl,[start]
Add dl,[end]
Sar dl,1
Mov bl,dl

Cmp al,[data+bx]
Je store
Ja store
Ja step1
Jb step2

Step1: add dl,1
      Mov [start],dl
Jmp loop1

Step2: add dl,1
      Mov [end],dl
Jmp loop1

Store: mov ax,1
mov ax , 0x4C00
int 0x21
End1: mov ax,0
          mov ax , 0x4C00
          int 0x21
```

Q) 64 bit multiplication

```
Mov cx,32
L1: shr word[num2+2],1
Rcr word[num2],1
Jnc skip
Mov ax,[num1]
Dd [result], ax
Mov ax,[num1+2]
Adc [result+2],ax
Mov ax,[num1+4]
Adc [result+4],ax
Mov ax,[num1+6]
Adc [result+6],ax
Skip:
Shl word[num1],1
Rcl word[num1+2],1
Rcl word[num1+4],1
Rcl word[num1+6],1
Dec cx
```

```
Jnz l1
mov ax , 0x4C00
        int 0x21
Num1: dq 0xABCDD4E1
num2 : dd 0xab5c32
Result: dq 0
```