Microprocessors Module 2 Questions

Part A

1. State significance of assembler directives in assembly language program, provide 2 examples

- Assembler directives help the assembler to correctly understand the assembly language programs to prepare the codes
- The following directives are commonly used in the assembly languag
  - DB - Define Byte The DB directive is used to reserve byte or bytes of memorylocations in the available memory.
  - ASSUME: Assume Logical Segment Name The ASSUME directive is used to inform the assembler, the names of the logical segments to be assumed for different segments used in the program
  - In the assembly language program, each segment is given aname. For example, the code segment may be given the name CODE, data segment may be given the name DATA etc.

2. List the 8086 Instructions used for transferring data between registers, memory, stack IO Devices

- MOV:
- PUSH
- POP
- PUSHA - Copy all registers to stack
- POPA - Copy words from stack to all registers

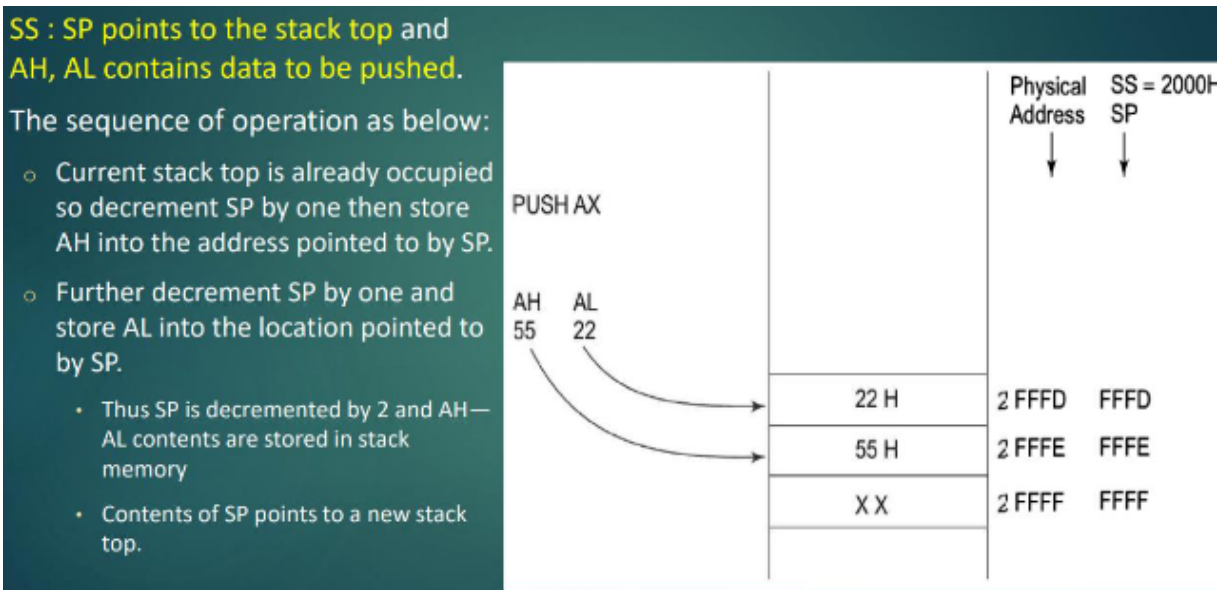
3. Write any three addressing mode of 8086 with example and write the effective address calculation in each

- Direct
  - Here 16-bit memory address (offset) is directly specified in the instruction
  - Example MOV AX, [5000 H]
  - Effective address is 10H * DS+5000H
    - Here the offset value is 5000
- Register Indirect

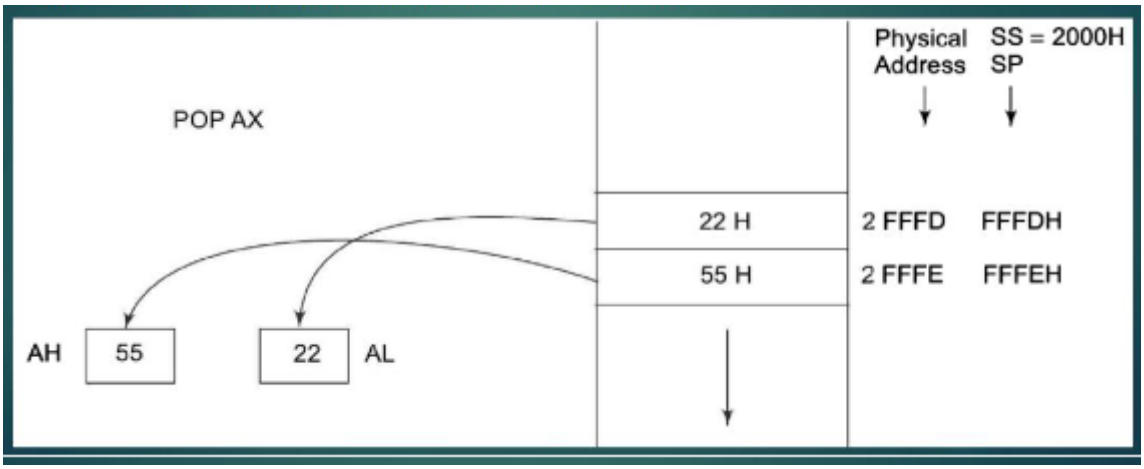
- In this mode, the offset address of data is in either BX or SI or DI register. The default segment is either DS or ES.
- Example: MOV AX, [BX]
- Effective address = 10H * DS+[BX]

- Indexed
  - In this addressing mode, offset of the operand is stored in one of the index registers.
  - Example: MOV AX, [SI]
  - The effective address is computed as 10H * DS+[SI]

4.Write the functions performed by PUSH and POP instructions in 8086 with appropriate diagram

- PUSH
  - Instruction pushes the contents of specified register/memory into the stack
    - Push decrements SP by 2
  - Example
    - PUSH AX
    - PUSH DS
    - PUSH [5000H]
      - Content of location 5000H and 5001H are pushed to stack
  - 

- POP
  - 16 bit contents of stack top are popped into specified operand
  - Content of stack top memory is stored in AL and SP is incremented by one
  - Further contents are copied to AH and SP is incremented by 1
  - POP loads the specified register/memory location with contents of memory location formed by stack segment and stack pointer

- Examples
  - POP AX
  - POP DS
  - POP [5000H]
- 

Part B

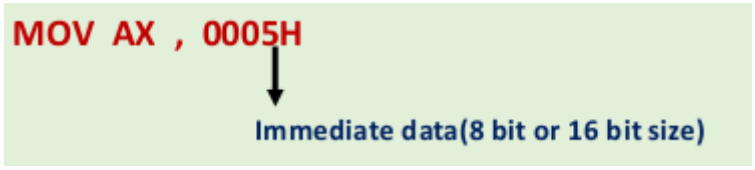
1. Discuss addressing modes supported by 8086 by suitable examples

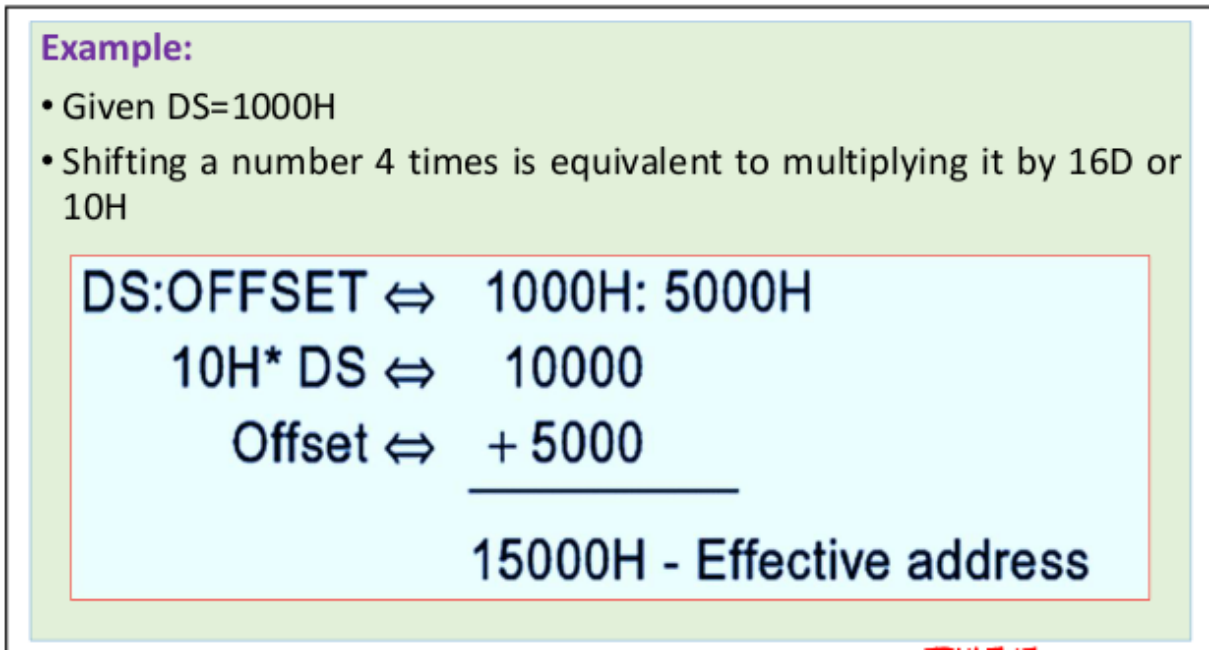
The different ways in which a source operand is denoted in an instruction is known as addressing modes.

- Types of instruction
  - Sequential control transfer instructions
    - Transfer control to next instruction immediately after it
    - Example Arithmetic , logical
  - Control transfer instructions
    - Transfer control to some predefined address/ address specified in the instruction
    - Example CALL, JUMP, RET

1. Addressing modes for sequential control transfer

- Immediate
  - Immediate data is part of instruction
  - immediate mode gives the computer the data right away, like handing over a number
  - Immediate data can be 8 bit or 16 bit

- 

- Direct
  - Memory address directly specified in the instruction
  - direct mode tells the computer where to find the data in its memory, similar to giving directions to a specific location.
  - **MOV AX , [5000H]**
  - Here the location is 5000H, The data inside 5000H Location is to be transferred to AX
  - • Effective address= offset address + segment address (content of DS)
    10H*DS+5000H
  - 

- Register
  - Data is stored in register. All the registers except IP can be used
  - Eg: **MOV BX, AX**
- Register Indirect
  - this instruction is saying, "Go to the memory location specified by the value in `BX`, get whatever is there, and put it into `AX`."
  - **MOV AX, [BX]**
  - **Effective address= 10H * DS+[BX]**

Example :

- Given DS=1000H and BX=2000H

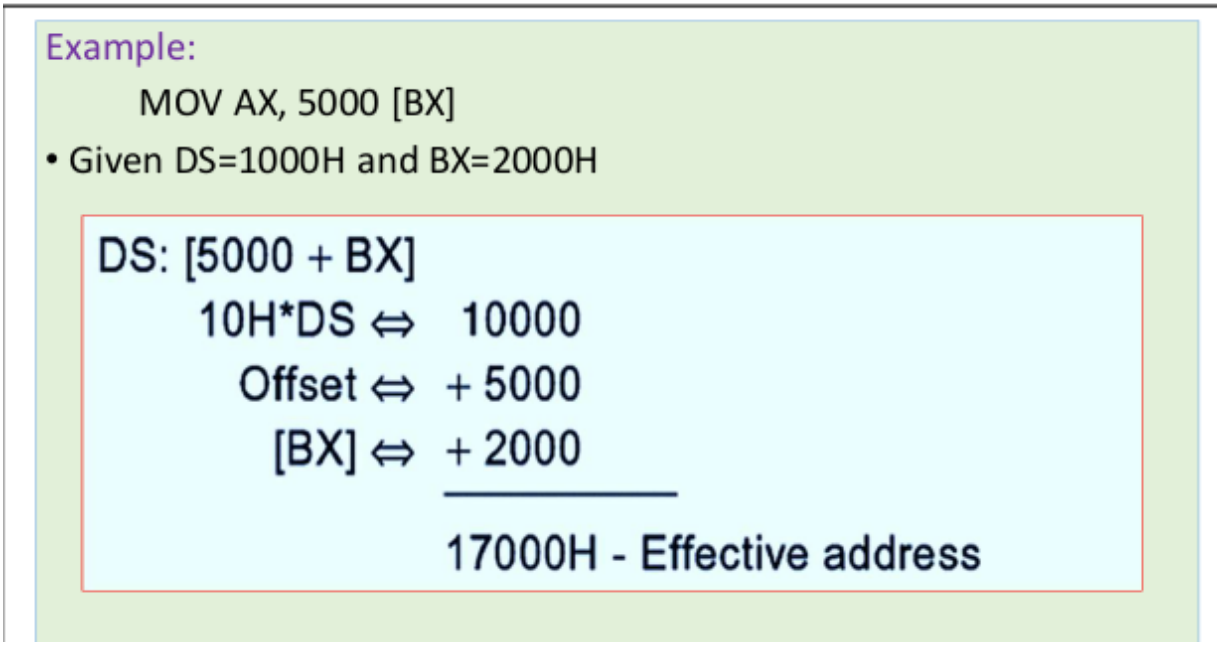
```
DS:BX ⇔  1000H:2000H
10H*DS ⇔   10000
   [BX] ⇔ + 2000
          ----------
          12000H - Effective address
```

- Indexed
  - offset of the operand is stored in one of the index registers.
  - For SI (Source index)
    - Default segment is DS
  - For DI (Destination index)
    - Default segment is ES

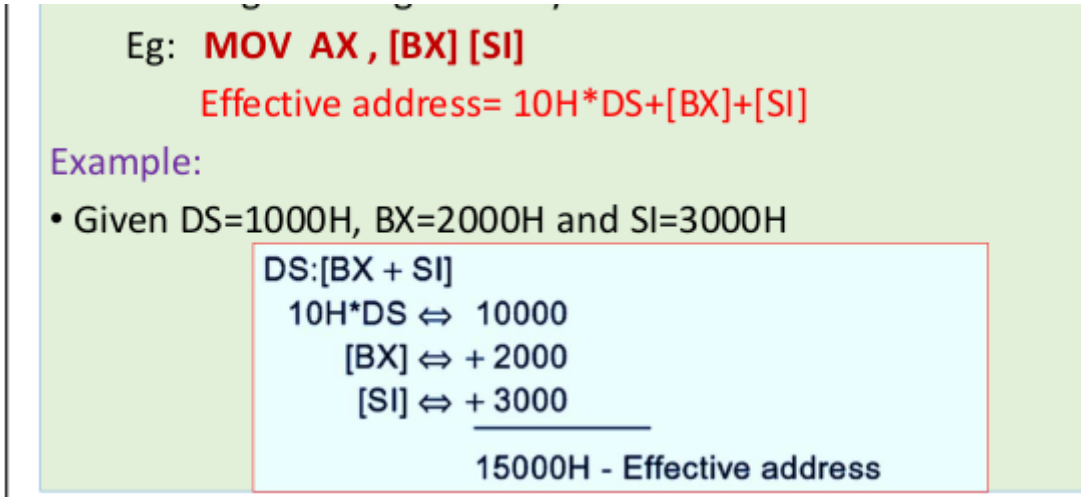
```
MOV  AX, [SI]
Effective address= 10H*DS+[SI]
```

- Register Relative
  - Data is available by adding the displacement with the content of any one of the register BX, BP, SI and DI

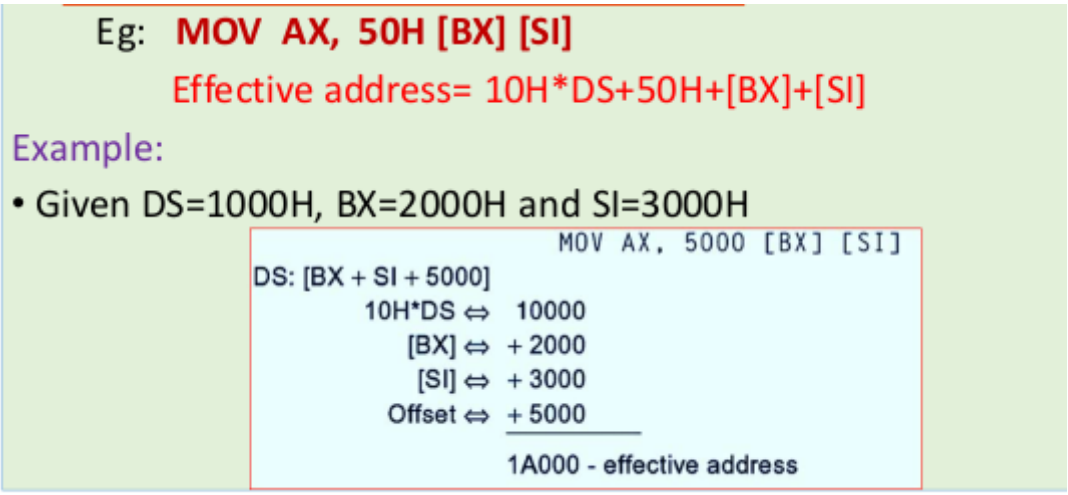
```
• Default segment is DS or ES
    Eg:  MOV  AX, 50H [BX]
         Effective address= 10H*DS+50H+[BX]
```



- Based Indexed
  - Effective address is sum of base register and Index register

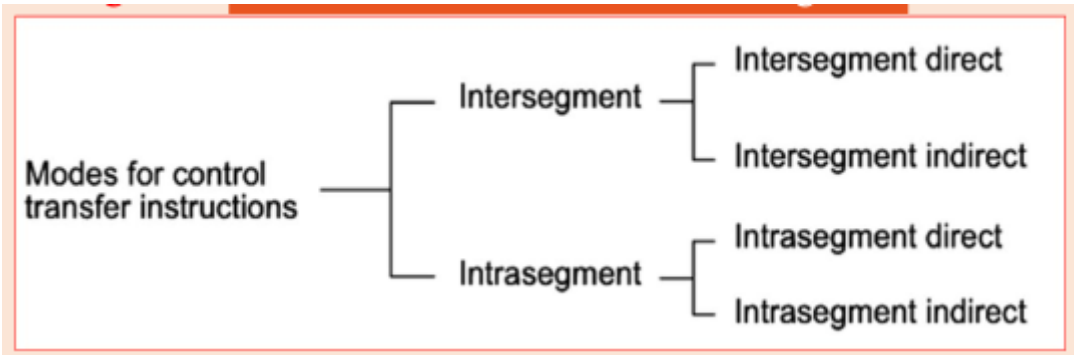


- Relative Based Indexed
  - effective address is formed by adding displacement with the sum of content of any of base registers (BX or BP) and any one of the index registers



2. Addressing modes for control transfer instruction

- Its classified into 2 types
  - Intersegment
  - Destination location is in different segment.
  - Intrasegment
  - Destination location is in same segment.



Intersegment Direct

- Destination location is in different segment.
- Provides branching from one code segment to another code segment
- Example

```
JPM 5000H : 2000H;
Jump to effective address 2000H in segment 5000H.
```

Intersegment Indirect

- Destination location is passed to the instruction indirectly

Example:

JMP [2000H];

- Jump to an address in the other segment specified at effective address 2000H in DS

Intrasegment Direct

- Displacement in computed using content of IP

Intrasegment InDirect

- Destination location is passed to the instruction indirectly.

Example

```
JMP [BX]; Jump to effective address stored in BX.
                                JMP [ BX + 5000H ]
```

2. Discuss about data transfer instructions with examples

Opcode	Operand	Description
MOV	D,S	Used to copy the byte or word from the provided source to the provided destination.
PUSH	D	Used to put a word at the top of the stack.
POP	D	Used to get a word from the top of the stack to the provided location.
PUSHA	----	Used to put all the registers into the stack.
POPA	----	Used to get words from the stack to all registers.
XCHG	D,S	Used to exchange the data from two locations.
IN	D,S	Used to read a byte or word from the provided port to the accumulator.
OUT	D,S	Used to send out a byte or word from the accumulator to the provided port.
XLAT	----	Used to translate a byte in AL using a table in the memory.

3. Assume that 8086 registers have values

- AX = 0030H
- BX=0031 H
- CX = 0032H
- DX = 0033h
- Flag - 000H
- Predict the values of Registers and Flags AX, BX,CX,DX,Carry Flag(CF), Zero Flag (ZF), Sign Flag(SF) after the execution of the following instructions,

(Assume each instruction is being executed independently)

- ROR AX,04h
- CMP BX,CX
- XCHG CX,DX
- AND AX,BX
- LOOP Addr
- XOR AX,AX
- STC

Flag)

Instruction	AX	BX	CX	DX	CF	ZF	SF
ROR AX,04H	0003H	0031H	0032H	0033H	0000H(If rotate with carry,CF is affected)	0000H	0000H
CMP BX,AX	0030H	0031H	0032H	0033H	If BX<AX CF is set	If BX=AX ZF is set	
XCHG CX,DX	0030H	0031H	0032H	0033H	0000H	0000H	0000H
AND AX,BX	0030H	0031H	0032H	0033H	0000H	0000H	0000H
LOOP Addr	0030H	0031H	0032H	0033H	0000H	0001H	0000H
XOR AX,BX	0001H	0031H	0032H	0033H	0000H	0001H	0000H
STC	0030H	0031H	0032H	0033H	0001H	0000H	0000H

4. Write an 8086 program to find largest among n numbers (Each numbers and count are of one byte only), Assume size of

array(count) stored in 2000h and numbers(array) stored from 2001h onwards upto n continues locations.

```
; Initialize SI with the memory address where the count of numbers (n) is
stored
4000 MOV SI, 2000

; Load the count of numbers (n) from memory into CL
4003 MOV CL, [SI]

; Initialize CH with 00
4005 MOV CH, 00

; Increment SI to point to the first number in the list
4007 INC SI

; Load the first number (AL) from the list
4008 MOV AL, [SI]

; Decrement the count of numbers (n)
400A DEC CL

; Increment SI to point to the next number in the list
400C INC SI

; Compare AL with the next number in the list ([SI])
400D CMP AL, [SI]

; Jump if not carry (JNC) to label 4013 if AL is less than or equal to the
next number
400F JNC 4013

; If AL is greater than the next number, update AL with the next number
4011 MOV AL, [SI]

; Increment SI to move to the next number in the list
4013 INC SI

; Loop back to the comparison (CMP) for the next pair of numbers
4014 LOOP 400D

; Store the largest number found in memory at address 3000
4016 MOV [3000], AL

; Halt the program
```

```
401A HLT
```

5. Write an assembly language program to find the largest and smallest number from an unordered array of 16-bit numbers. Assume the array contains 15 numbers and the starting location as 2500H. Draw the flowchart for the program.

```
MOV AX, 0 ; Initialize AX as the largest number
MOV BX, 65535 ; Initialize BX as the smallest number
MOV CX, 15 ; Set the counter to the number of elements
MOV SI, 2500H ; Set the starting location of the array
LOOP_START:
MOV DX, [SI] ; Load the current element into DX
CMP DX, AX ; Compare with the largest number
JG UPDATE_MAX ; Jump if DX > AX
CMP DX, BX ; Compare with the smallest number
JL UPDATE_MIN ; Jump if DX < BX
JMP NEXT_ELEMENT ; Jump to the next element
UPDATE_MAX:
MOV AX, DX ; Update AX with the new largest number
JMP NEXT_ELEMENT ; Jump to the next element
UPDATE_MIN:
MOV BX, DX ; Update BX with the new smallest number
NEXT_ELEMENT:
ADD SI, 2 ; Move to the next element (16-bit)
LOOP LOOP_START ; Repeat until all elements are processed
; After the loop, the largest number will be in AX, and the
smallest number will be in BX
```

6. Write an assembly language program to find the total number of even and odd numbers from an array of 16-bit numbers. Assume the array contains 20 numbers and the starting location as 5500H. Draw the flowchart for the program.

```
; Define the segments and assume the code and data segments
ASSUME CS:CODE, DS:DATA

DATA SEGMENT
    ; Define an array of 16-bit numbers
    LIST DW 2357H, 0a579H, 0c2322H, 0c91eH, 0c0000H, 0957H
    ; Define the count of numbers in the array
```

```
    COUNT EQU 006h
DATA ENDS

CODE SEGMENT
START:
    ; Initialize BX and DX to 0 for counting even and odd numbers
    XOR BX, BX
    XOR DX, DX

    ; Load the data segment address into AX and DS
    MOV AX, DATA
    MOV DS, AX

    ; Load the count of numbers into CL
    MOV CL, COUNT

    ; Initialize SI to the offset of the array LIST
    MOV SI, OFFSET LIST

AGAIN:
    ; Load the current number from the array into AX
    MOV AX, [SI]

    ; Check if the number is even
    TEST AX, 01
    JZ EVEN

    ; If odd, increment the odd counter (DX) and jump to NEXT
    INC DX
    JMP NEXT

EVEN:
    ; If even, increment the even counter (BX)
    INC BX

NEXT:
    ; Move to the next number in the array
    ADD SI, 02

    ; Decrement the count of remaining numbers
    DEC CL

    ; If there are more numbers, jump back to AGAIN
    JNZ AGAIN

    ; Set up the exit code
```

```
    MOV AH, 4CH
    INT 21H

CODE ENDS
END START
```