0. Write a sequence of instructions (that is, not a full program with TITLE, ASSUME, etc.) that checks whether the memory in a full 64KB segment is functioning correctly. Assume that register AX contains the segment to be checked. Use the string instruction STOSB to load the byte AAH into all 64K locations, and then use LODSB to check that AAH was correctly stored at each of the locations. If the wrong value is read, then call a fictitious subroutine MemError. Otherwise, call Function 4CH of DOS INT 21H to exit.

; Instruction sequence to check 64KB of memory using STOSB and LODSB
; Initialize
    MOV DS, AX ; Set segments
    MOV ES, AX
    CLD ; Set direction flag for increments

; Store 64K byte values to memory
    MOV DI, 0 ; Set starting offset
    MOV AL, 0AAh ; Set test value
    MOV CX, 0FFFFh ; Set for 64K-1 locations
    REP STOSB ; Store test value
    STOSB ; Extra store for total of 64KB

; Check that values have been stored correctly
    MOV SI, 0 ; Set starting offset
    MOV AH, 0AAh ; Set compare pattern
    MOV CX, 0 ; Set maximum loops (64KB)
    Again: LODSB ; Load value from memory
            CMP AH, AL ; Is value valid?
            JNE HandleError ; No, there is an error in memory.
            LOOP Again ; Continue comparing
            JMP Exit ; All loaded values are valid

HandleError: CALL MemError ; Error, call subroutine.
Exit: INT 6

Notes:

I stated to at least one person that loading register CX with 0 before execution of the REP STOSB instruction would provide for automatically storing to 65536 (64K) locations. However, that is not the case; if you try the above program in DEBUG with CX set to 0, then STOSB will not execute at all. This corroborates the description of REP in the Intel Handbook, which states that REP adds a do while structure to string operation. On the other hand, LOOP functions differently. It first decrements CX, and then checks to see if it is 0. If not, it jumps. Thus if CX is 0, then LOOP will jump 65536 times, as required by the question.

In your implementation, you do not need to call INT 6 after you call the subroutine.
1. Read Intel 8086/88 User Manual page 6-53 to 6-55: understand how the instruction is encoded to and decoded from machine code. Use the two tables that followed (table 6-22 and 6-23); do the following two questions. Convert the following hexadecimal machine codes to assembly language mnemonics. State what each of the byte fields mean (Table 6-23 from page 6-64 to page 6-69).

a. B8 00 20  
   B8 = MOV AX, IMMED16  
   00 = Data-lo  
   20 = Data-hi  
   Answer: MOV AX, 2000h

b. 8E D8  
   8E = MOV SEGREG, REG16 / MEM16  
   D8 = 2nd byte: MOD 0 SR R/M  
   MOD = 11 Register Mode  
   SR = 11 Segment register DS (see page 3-57) – operand 1  
   R/M = 000 Register AX – operand 2  
   Answer: MOV DS, AX

c. 46  
   46 = INC SI  
   Answer: INC SI

d. 90  
   90 = NOP  
   Answer: NOP
e. \( 89 \ 7C \ \text{FE} \)

<table>
<thead>
<tr>
<th>Hex</th>
<th>Opcode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>89</td>
<td>MOV</td>
<td>REG16 / MEM16, REG16 (note error in table 6-23)</td>
</tr>
<tr>
<td>7C</td>
<td>2nd byte:</td>
<td>MOD REG R/M</td>
</tr>
<tr>
<td>FE</td>
<td>8-bit signed displacement</td>
<td></td>
</tr>
</tbody>
</table>

MOD = 01 Memory Mode, 8-bit displacement follows
REG = 111 DI – operand #2
R/M = 100 (SI) + D8 - operand #1

FE = 8-bit signed displacement

\[
\begin{array}{c}
\text{11111110} \\
\text{2's complement: 00000010 = 2}
\end{array}
\]

Thus the displacement is \(-2\)

Answer: \( \text{MOV [SI]-2, DI} \)

f. \( 75 \ \text{F7} \)

<table>
<thead>
<tr>
<th>Hex</th>
<th>Opcode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>75</td>
<td>JNE/JNZ short-label</td>
<td></td>
</tr>
<tr>
<td>F7</td>
<td>IP – INC8 (8-bit signed offset to add to IP)</td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{array}{c}
\text{11101111} \\
\text{2's complement: 00001001 = 9}
\end{array}
\]

Thus the offset is \(-9\)

Answer: \( \text{JNE -9} \) (that is, jump back 9 machine code bytes if not equal) or \( \text{JNZ -9} \)

g. \( \text{E2} \ \text{EF} \)

<table>
<thead>
<tr>
<th>Hex</th>
<th>Opcode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E2</td>
<td>LOOP short-label</td>
<td></td>
</tr>
<tr>
<td>EF</td>
<td>IP – INC8 (8-bit signed offset to add to IP)</td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{array}{c}
\text{11101111} \\
\text{2's complement: 00010001 = 17}
\end{array}
\]

Thus the offset is \(-17\)

Answer: \( \text{LOOP –17} \) (loop back 17 machine bytes)

h. \( \text{26} \ \text{80} \ 07 \ \text{78} \)

<table>
<thead>
<tr>
<th>Hex</th>
<th>Opcode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>Segment override – ‘ES:’</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>One of several choices; look at bits 3, 4, &amp; 5 of next byte</td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>Bits 3, 4, &amp; 5 are ‘000’, so instruction is</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADD REG8 / MEM8, IMMED8</td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>8-bit immediate value</td>
<td></td>
</tr>
</tbody>
</table>

Note: instruction must have BYTE PTR to indicate an 8-bit operation

Answer: \( \text{ADD BYTE PTR ES:[BX], 78h} \)
2. Convert the following instructions to machine code – give your answers in hexadecimal. State what each of the bit fields mean.
   i. PUSH BX
   j. MOV [SI+490], SP
   k. OUT DX, AL
   l. POPF
   m. AND AX, [BX+DI+2Dh]
   n. ADD DS:[BP], DX  Note: you will have to ‘add’ a displacement
   o. XOR AL, [BX+DI-36H]
   p. MOV [DI+476], ES

a. PUSH BX – two possible answers

Memory or Register Operand

<table>
<thead>
<tr>
<th></th>
<th>mod 110 r/m</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mod = 11</td>
</tr>
<tr>
<td></td>
<td>r/m = 011</td>
</tr>
</tbody>
</table>

Answer #1: FF F3

Register Operand

<table>
<thead>
<tr>
<th>reg</th>
</tr>
</thead>
<tbody>
<tr>
<td>reg = 011</td>
</tr>
</tbody>
</table>

Answer #2: 53

b. MOV [SI+490], SP

Memory or Register Operand to/from Register Operand

<table>
<thead>
<tr>
<th></th>
<th>mod reg r/m</th>
<th>disp-lo</th>
<th>disp-hi</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>d = 0</td>
<td>w = 1</td>
<td>mod = 10</td>
</tr>
<tr>
<td></td>
<td>reg = 100</td>
<td>r/m = 100</td>
<td>EA = (SI) + D16</td>
</tr>
<tr>
<td></td>
<td>disp = 1EAh</td>
<td>490 = 1EAh</td>
<td></td>
</tr>
</tbody>
</table>

Answer: 89 A4 EA 01

c. OUT DX, AL

Variable Port

<table>
<thead>
<tr>
<th>w</th>
</tr>
</thead>
<tbody>
<tr>
<td>w = 0</td>
</tr>
</tbody>
</table>

Answer: EE
d. \text{POPF} \\
\begin{array}{c}
0011101
\end{array} \\
\text{Answer: 9D}

e. \text{AND AX, [BX+DI+2Dh]} \\
\text{Memory or Register Operand with Register Operand} \\
\begin{array}{ccc}
010000 & d & w \\
& \text{mod reg r/m} & \text{disp-lo}
\end{array} \\
d = 1 \quad \text{To register} \\
w = 1 \quad \text{Word operands (AX is 2 bytes large)} \\
\text{mod} = 01 \quad \text{Memory Mode, 8-bit displacement follows (2Dh)} \\
\text{reg} = 000 \quad \text{Register AX} \\
\text{r/m} = 001 \quad (BX) + (DI) + D8 \\
\text{disp} = 2D \quad \text{Displacement is 2Dh, which can fit in a 1 byte signed number} \\
\text{Answer: 23 41 2D}

f. \text{ADD DS:[BP], DX} \\
\text{Segment override: It is a bit tricky finding the prefix byte for segment overrides. If you look in Table 6-22 of the Intel User’s Manual, on page 6-61, you will see that the last entry is:} \\
\text{SEGMENT=Override prefix} \begin{array}{c}
0011101
\end{array} \\
\text{The \text{reg} field is actual a segment register field, and you can use the \text{Segment} column of the \text{“reg” Field Bit Assignments} chart on page 3-57 to determine how to set it.} \\
\text{Segment DS Override:} \begin{array}{c}
0011110
\end{array} = 3E \\
\text{Note: you will have to ‘add’ a displacement} \\
\text{The addition of a displacement to the memory reference (that is, [BP] ), is needed because there is no encoding for [BP]. Logically, the \text{mod} field should be 00, and the \text{r/m} field should be 110. But that is a special case, for when a direct address is used (something like [1000h]). To encode, you will have to rewrite the instruction into the functionally equivalent form:} \\
\text{ADD DS:[BP+0], DX} \\
\text{Memory or Register Operand with Register Operand} \\
\begin{array}{ccc}
000000 & d & w \\
& \text{mod reg r/m} & \text{disp-lo}
\end{array} \\
d = 0 \quad \text{From register} \\
w = 1 \quad \text{Word operands (DX is 2 bytes large)} \\
\text{mod} = 01 \quad \text{Memory Mode, 8-bit signed displacement follows} \\
\text{reg} = 010 \quad \text{Register DX} \\
\text{r/m} = 110 \quad (BP) + D8 \\
\text{disp} = 00 \quad \text{Displacement is 0} \\
\text{Answer: 3E 01 56 00}

g. \text{XOR AL, [BX+DI-36H]} \\
\text{Memory or Register Operand with Register Operand} \\
\begin{array}{ccc}
001100 & d & w \\
& \text{mod reg r/m} & \text{disp-lo}
\end{array} \\
d = 1 \quad \text{To register} \\
w = 0 \quad \text{Byte operands (AL is 1 byte large)} \\
\text{mod} = 01 \quad \text{Memory Mode, 8-bit displacement follows} \\
\text{reg} = 000 \quad \text{Register AL} \\
\text{r/m} = 001 \quad (BX) + (DI) + D8 \\
\text{disp} \quad \text{Displacement is –36h, which can fit in an 8-bit 2’s complement form}
6h = 0011 0110b -> (2’s comp) -> 11001010b = CAh = disp

Answer: 32 41 CA

h. MOV [DI + 476], ES

Segment Register to Memory or Register Operand

<table>
<thead>
<tr>
<th>mod = 10</th>
<th>Memory Mode, 16-bit displacement follows</th>
</tr>
</thead>
<tbody>
<tr>
<td>reg = 00</td>
<td>Register ES (note that this is a segment register, and thus is 2 bits large)</td>
</tr>
<tr>
<td>r/m = 101</td>
<td>(DI) + D16</td>
</tr>
<tr>
<td>disp = 01DC</td>
<td>Displacement is 47610 = 1DC16</td>
</tr>
</tbody>
</table>

Answer: 8C 85 DC 01

3. The following bytes are found in order somewhere in memory. Assuming they are machine codes, decode the values into meaningful assembly language mnemonics.

B9 00 12 D0 C0 E8 C8 E2 F9

B9 – MOV CX, ImmeD16

The next two bytes are the immediate 16-bit value loaded into CX (00 12 -> 1200H)

MOV CX, 1200H

D0 – One of eight possibilities (ROL, ROR, RCL, etc.), so use the next byte.

C0 = 11000000. The first two digit (MSB) “11” is MOD and “11” means that r/m = reg field. The next three digits “000” indicates that ROL Reg8, 1. The last three digits (LSB) is “000”, it is R/M field and represent AL.

ROL AL, 1

E8 – CALL Near-proc

Indicates that there is a call to a subroutine in the same segment. The next two bytes (IP-INC-Lo and IP-INC-Hi) give a signed 16-bit displacement from the current value of IP.

Disp = E2C8 (negative)

= 1110001011001000b -> -000111010011100b = -1D38H = -7480D

CALL [IP-7480]

F9 – STC

4. Use full segment definition, write a DOS compatible program that: a) clears the screen, b) set the cursor to screen position row = 10 and column = 5, c) displays the prompt “Please enter an 8-digit number: “, d) get the keyboard input and save the number to a buffer area in the memory (you define), e) sort the number on its ascending order and save them to another buffer for display. For example, if the input number is 29034765, then after your sort, the result should be 02345679. You can assume that the number for each digit is non-repeat but actually the repeated case is the same, f) after your sort, change to the start of next new line, output “The sorted number is: ” and the number, g) exit use DOS function 4CH. Write task a) and b) using subroutines. Test your code on PC by yourself.

;Tasks:
;(1) Clear the screen use subroutine
;(2) Set the cursor to ROW 10 and COLUMN 5 on the screen use subroutine
;(3) Output a prompt string: "Please enter an 8-digit number:" 
;(4) Accept keyboard input: (put to buffer INPUT_BUF)
;(5) Sort the number on its ascending order
;(6) move the sorted number to display buffer (OUTPUT_BUF)
;(7) Change to a new line and Output string: "The sorted number is :" and the number and then exit
TITLE Example
PAGE  120, 60

;-------------------------------
LF EQU 0DH
CR EQU 0AH
;-------------------------------
STSEG SEGMENT
DB 64 DUP(?)
STSEG ENDS
;-------------------------------
DTSEG SEGMENT
PROMPT1  DB 'Please enter an 8-digit number:','$'
PROMPT2  DB LF,CR,'The sorted number is : ',LF,CR,'$'

INPUT_BUF LABEL BYTE
SIZE_IBUF DB 09H
INUMBER  DB 00H
INPUT_DATA DB 9 DUP(0FFH)

DTSEG ENDS
;-------------------------------
CDSEG SEGMENT
MAIN PROC FAR
ASSUME CS:CDSEG, DS:DTSEG, SS:STSEG, ES:DTSEG
MOV AX, DTSEG
MOV DS, AX
MOV ES, AX
CALL CLEAR
CALL CURSOR

MOV AH, 09H ;Prompt digit inputs
MOV DX, OFFSET PROMPT1
INT 21H

MOV AH, 0AH ;get inputs
MOV DX, OFFSET INPUT_BUF
INT 21H

MOV BX, 7 ;sort the number using natural sort method
MOV SI, OFFSET INPUT_DATA
REPO:  MOV DI, SI
       ADD DI, 1

MOV CX, BX
REP1:  MOV AL, [SI]
       MOV AH, [DI]
       CMP AH, AL
       JA CONT0

       MOV [SI], AH
       MOV [DI], AL

CONT0: INC DI
       LOOP REP1

       INC SI
       DEC BX
       JNZ REP0

CDSEG ENDS
MOV AH, 09H ; send the output prompt
MOV DX, OFFSET PROMPT2
INT 21H

MOV DX, OFFSET INPUT_DATA ; prepare for the sorted sequence output
MOV BX, DX
MOV BYTE PTR [BX+8], '$'
MOV AH, 09H ; output the sorted sequence
INT 21H

MOV AH, 4CH
INT 21H

MAIN ENDP

;--------------------------------CLEAR THE SCREEN
CLEAR PROC NEAR
MOV AH, 06
MOV AL, 00
MOV BH, 07
MOV CX, 0000
MOV DX, 184FH
INT 10H
RET
CLEAR ENDP

;--------------------------------MOVE CURSOR TO THE POSITION
CURSOR PROC NEAR
MOV AH, 02
MOV BH, 00
MOV DL, 00 ; columns
MOV DH, 00 ; rows
INT 10H
RET
CURSOR ENDP

;-------------------------------
CDSEG ENDS
END MAIN