ISLAMIC UNIVERSITY OF TECHNOLOGY(IUT)
ORGANIZATION OF ISLAMIC COOPERATION
DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

Final Assignment

Name : Fariha Alam Urbana
Student id : 200021315
Section : C ,C1

Course no : EEE 4616
Course Title : Microprocessor and Assembly Language

Programming Lab

This Problem includes Taking input from user ,sorting ,searching algorithm with the help of assembly language in EMU 8086.It asked for
Task1)Composing an assembly program that receives input for item names and corresponding prices
sequentially, storing the information.
Task2) Then arrange the prices in ascending order and display the complete list of items with their names and sorted prices.and
task3)finally to employ any search algorithm to find out all the items' names with price more than 20$.

The whole problem is divided into three parts

1. **Input Processing:** We need to create an assembly program that takes input for item names and corresponding prices sequentially and stores this information efficiently.
2. **Sorting Algorithm:** After receiving all the input, we need to write an assembly program to arrange the prices in ascending order. This means implementing a sorting algorithm to reorder the prices while keeping track of their corresponding item names.
3. **Search Algorithm:** Once the items and their sorted prices are available, we need to employ a search algorithm to find all the items with prices greater than $20.

Part 1: Input Processing

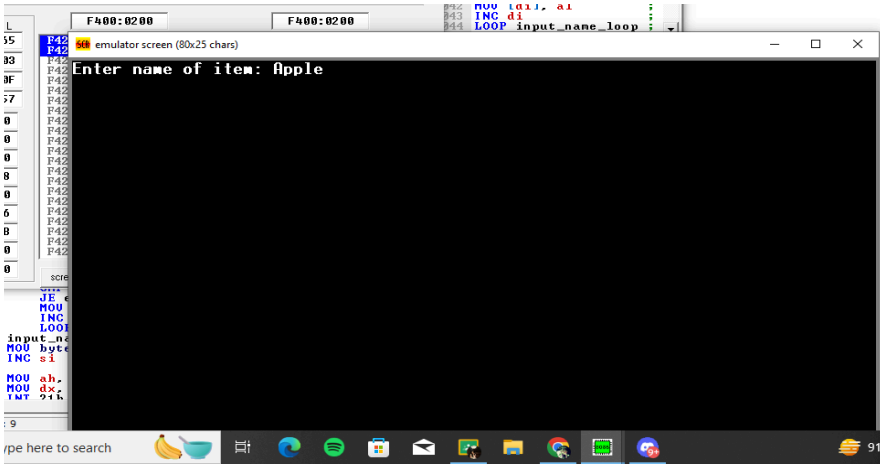
```
include 'emu8086.inc'

org 100h

.model small

.data
    item_no equ 3                ; Maximum number of items
    arr db item_no dup(?)        ; Holds the prices of the items
    item_names db item_no*20 dup(?)  ; Holds the names of the items (assuming each name is maximum 20 characters long)
    len equ ($-arr)/1
    key equ 3
    failmsg db 'Key is not found!!!.$'  ; Message to print if key is not found
    input_name_prompt db "Enter name of item: $"
    input_price_prompt db "Enter price of item: $"

.code


input_proc:
    MOV cx, item_no      ; Set to determine the iteration of the input loop
    MOV bx, offset arr  ; Initialize index for arr
    MOV si, offset item_names  ; Initialize index for item_names

input_loop:

    MOV ah, 09h               ; DOS function to print string
    MOV dx, offset input_name_prompt
    INT 21h

    ; Input item name
    MOV ah, 0Ah                  ; DOS function for buffered input
    MOV di, si                      ; Store current index of item_names in di
    MOV cx, 20                ; Maximum characters for item name
    MOV ah, 01h              ; Function to input a character with echo
    input_name_loop:
        INT 21h                ; Call DOS to input a character
        CMP al, 13             ; Check for Enter key
        JE input_name_done    ; If Enter key is pressed, exit the loop
        CMP al, '0'                 ; To Check if the input is '0'
        JE exit_input_proc     ; If '0' is pressed, exit the input procedure
        MOV [di], al           ; Store the input character in item_names
        INC di                 ; Move to the next character in item_names
        LOOP input_name_loop ; Repeat until maximum characters are reached
    input_name_done:
    MOV byte ptr [di], '$'    ; Append '$' to mark end of string
    INC si

    MOV ah, 09h               ; DOS function to print string
    MOV dx, offset input_price_prompt
    INT 21h
```

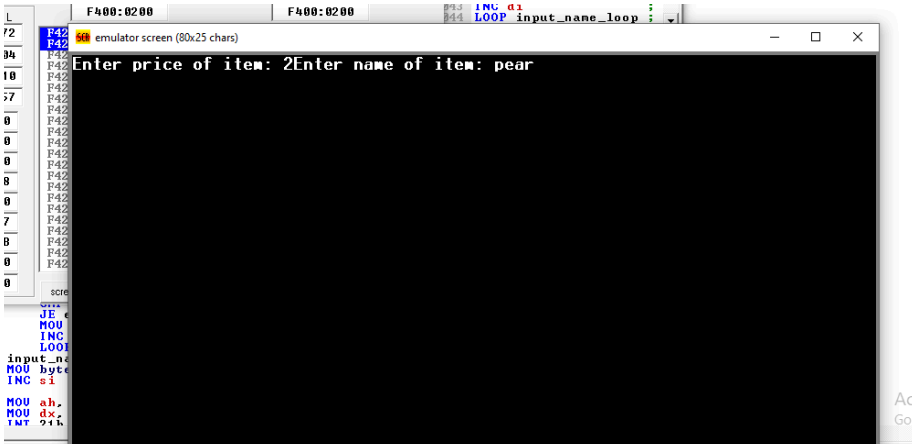
new open examples save | compile emulate | calculator convertor | options help about

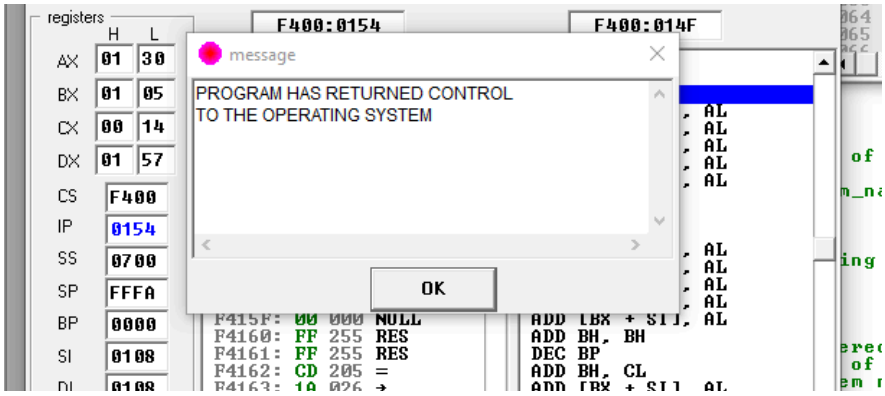
```
    MOV ah, 09h               ; DOS function to print string
    MOV dx, offset input_price_prompt
    INT 21h

    ; Input item price
    MOV ah, 01h                ; DOS function for input with echo
    INT 21h                   ; Call DOS to input the item price
    SUB al, '0'                ; Convert ASCII to binary
    MOV [bx], al              ; Store the price in arr
    INC bx

    DEC cx                     ; Decrement the counter for the number of items

    JMP input_loop

exit_input_proc:
    RET

MAIN PROC
    MOV ax, @data
    MOV ds, ax

    CALL input_proc  ; Call the procedure to input item names and prices

    ; Print the inputted item names and prices
    MOV cx, item_no
    MOV bx, 0
    print "Inputted item names and prices:"

    Outputs:
        MOV ah, 09h
        MOV dx, offset item_names + bx*20 ; Load address of item name
        INT 21h           ; Print item name
        MOV dl, ','        ; Print delimiter
        MOV ah, 02h     ; Print character function
        INT 21h
        MOV al, arr[bx]  ; Load price
        ADD al, '0'          ; Convert to ASCII
        MOV dl, al        ; Move ASCII to dl
        MOV ah, 02h        ; Print character function
        INT 21h
        INC bx             ; Move to next item
        LOOP Outputs     ; Loop if not all items are printed

    ; New line
    MOV dl, 10
    MOV ah, 02h
    INT 21h

    MOV dl, 13
```

new open examples save compile emulate calculator convertor options help about

```
MAIN PROC
    MOV ax, @data
    MOV ds, ax

    CALL input_proc  ; Call the procedure to input item names and prices

    ; Print the inputted item names and prices
    MOV cx, item_no
    MOV bx, 0
    print "Inputted item names and prices:"

    Outputs:
        MOV ah, 09h
        MOV dx, offset item_names + bx*20 ; Load address of item name
        INT 21h                 ; Print item name
        MOV dl, ','             ; Print delimiter
        MOV ah, 02h         ; Print character function
        INT 21h
        MOV al, arr[bx]   ; Load price
        ADD al, '0'               ; Convert to ASCII
        MOV dl, al          ; Move ASCII to dl
        MOV ah, 02h         ; Print character function
        INT 21h
        INC bx                  ; Move to next item
        LOOP Outputs      ; Loop if not all items are printed

    ; New line
    MOV dl, 10
    MOV ah, 02h
    INT 21h

    MOV dl, 13
    INT 21h

    .

    main endp

ret
```







Input Processing:

1. **Initialization:**
    - `MOV cx, item_no`: Sets the loop counter `cx` to the maximum number of items.
    - `MOV bx, offset arr`: Initializes the index `bx` to point to the beginning of the `arr` array.
    - `MOV si, offset item_names`: Initializes the index `si` to point to the beginning of the `item_names` array.
2. **Input Loop:**
    - This loop iterates for each item.

- It prompts the user to enter the name of the item (`input_name_prompt`).
- Using `INT 21h`, it reads the inputted item name character by character until the Enter key is pressed.
- Each character is stored in the `item_names` array.
- Once the name is entered, it appends a '$' character to mark the end of the string.
- Then, it prompts the user to enter the price of the item (`input_price_prompt`).
- It reads the inputted price character by character, converts ASCII to binary, and stores it in the `arr` array.
- The loop continues until all items are processed.

3. **Loop Termination and Exit:**
   - `DEC cx`: Decrements the loop counter to keep track of the number of items remaining.
   - `JMP input_loop`: Jumps back to the beginning of the input loop to process the next item.
   - `exit_input_proc: RET`: Marks the end of the input processing procedure.

This part of the code efficiently handles the input of item names and prices from the user. It iterates through each item, prompting the user to enter the name and price, and stores this information in the respective arrays (`item_names` and `arr`). It ensures the input process is terminated correctly and efficiently.

Part 2:Sorting

I have used bubble sorting algorithm for sorting the prices into ascending order.
The algorithm iterates through the array multiple times, comparing adjacent elements.

```
.model small
.stack 100h

.data
    ARR db 9 DUP(?)
    COUNT dw 7
    MSG db 10, 13, "Enter ", COUNT, " prices of items: $"
    NEWLINE db 10, 13, "$"

.code
    mov ax, @data
    mov ds, ax

    ; Input loop (read COUNT integers)
    mov cx, COUNT
    mov si, 0
input_loop:
    lea dx, MSG
    mov ah, 09h
    int 21h

    mov ah, 01h
    int 21h
    sub al, 30h ; Convert ASCII to integer
    mov ARR[si], al

    inc si
    loop input_loop

    ; Bubble sort
    mov cx, COUNT
    dec cx
outer_loop:
    mov bx, cx
    mov si, 0
inner_loop:
    mov al, ARR[si]
    mov dl, ARR[si + 1]
    cmp al, dl
    jbe no_swap

    ; Swap elements
    mov al, ARR[si]
    mov ah, ARR[si + 1]
    mov ARR[si], ah
    mov ARR[si + 1], al

no_swap:
    inc si
    dec bx
    jnz inner_loop
```

```
    mov al, ARR[si]
    mov ah, ARR[si + 1]
    mov ARR[si], ah
    mov ARR[si + 1], al

no_swap:
    inc si
    dec bx
    jnz inner_loop

    dec cx
    jnz outer_loop

    ; Display original elements
    mov cx, 7
    mov si, 0
print_original:
    mov al, ARR[si]
    add al, 30h
    mov ah, 0eh
    int 10h
    mov dl, ' '
    int 21h
    inc si
    loop print_original

    ; Display newline
    lea dx, NEWLINE
    mov ah, 09h
    int 21h

    ; Display sorted elements
    mov cx, 7
    mov si, 0
print_sorted:
    mov al, ARR[si]
    add al, 30h
    mov ah, 0eh
    int 10h
    mov dl, ' '
    int 21h
    inc si
    loop print_sorted

    ; Exit
    mov ah, 4Ch
    int 21h

end
```

```
emulator screen (80x25 chars)

Enter    prices of items: 3
Enter    prices of items: 4
Enter    prices of items: 8
Enter    prices of items: 9
Enter    prices of items: 2
Enter    prices of items: 0
Enter    prices of items: 10123489
0123489
```

Input Section:

- The program prompts the user to enter a specified number of prices for items. This number is determined by the `COUNT` variable.
- It reads each entered price character by character, converts it from ASCII to an integer, and stores it in the `ARR` array.

Bubble Sort Algorithm:

- The program implements the bubble sort algorithm to sort the entered prices in ascending order.
- The outer loop iterates through the array from the end towards the beginning. It controls the number of iterations required to sort the entire array.
- Within the outer loop, the inner loop compares adjacent elements in the array and swaps them if they are in the wrong order.
- If no swaps are made in an iteration of the inner loop, it means that the array is already sorted, and the outer loop terminates early.

Display Section:

- After sorting the array, the program displays the original prices followed by a newline character.
- Then, it displays the sorted prices, both separated by spaces.

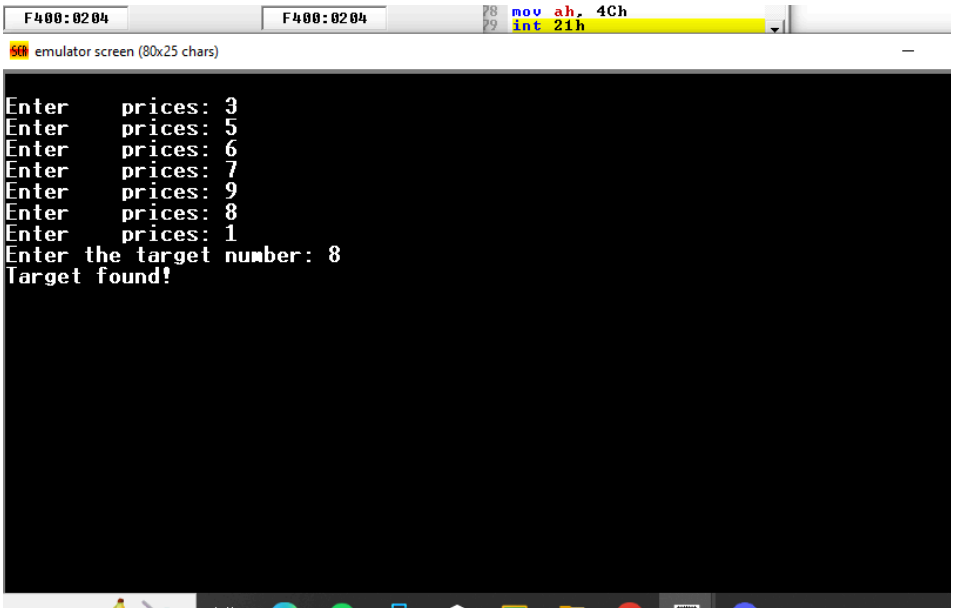
Part 3: Searching

For searching prices above $20i have implemented binary search algorithm .

```
.model small
.stack 100h

.data
    ARR db 9 DUP(?)
    COUNT dw 7
    MSG db 10, 13, "Enter ", COUNT, " prices: $"
    TARGET db 10, 13, "Enter the target number: $"
    FOUND_MSG db 10, 13, "Target found!$"
    NOT_FOUND_MSG db 10, 13, "Target not found!$"

.code
    mov ax, @data
    mov ds, ax

    ; Input loop (read COUNT integers)
    mov cx, COUNT
    mov si, 0
input_loop:
    lea dx, MSG
    mov ah, 09h
    int 21h

    mov ah, 01h
    int 21h
    sub al, 30h ; Convert ASCII to integer
    mov ARR[si], al

    inc si
    loop input_loop

    ; Input target number
    lea dx, TARGET
    mov ah, 09h
    int 21h

    mov ah, 01h
    int 21h
    sub al, 30h ; Convert ASCII to integer
    mov bl, al ; Store the target number

    ; Binary search
    mov cx, COUNT
    mov si, 0
search_loop:
    mov bx, si
    add bx, cx
    shr bx, 1 ; Calculate mid index

    mov al, ARR[bx]
    cmp al, bl
```

new open examples save compile emulate calculator convertor options help about

```
    mov ah, 09h
    int 21h

    mov ah, 01h
    int 21h
    sub al, 30h ; Convert ASCII to integer
    mov bl, al ; Store the target number

    ; Binary search
    mov cx, COUNT
    mov si, 0
search_loop:
    mov bx, si
    add bx, cx
    shr bx, 1 ; Calculate mid index

    mov al, ARR[bx]
    cmp al, bl
    je target_found
    jl move_right
    jg move_left

move_right:
    mov si, bx
    inc si
    loop search_loop

move_left:
    mov cx, bx
    loop search_loop

target_found:
    lea dx, FOUND_MSG
    mov ah, 09h
    int 21h
    jmp exit_program

not_found:
    lea dx, NOT_FOUND_MSG
    mov ah, 09h
    int 21h

exit_program:
    ; Exit
    mov ah, 4Ch
    int 21h

end
```



Input Section:

- The program prompts the user to enter a specified number of prices for items (`COUNT`).
- It reads each entered price character by character, converts it from ASCII to an integer, and stores it in the `ARR` array.

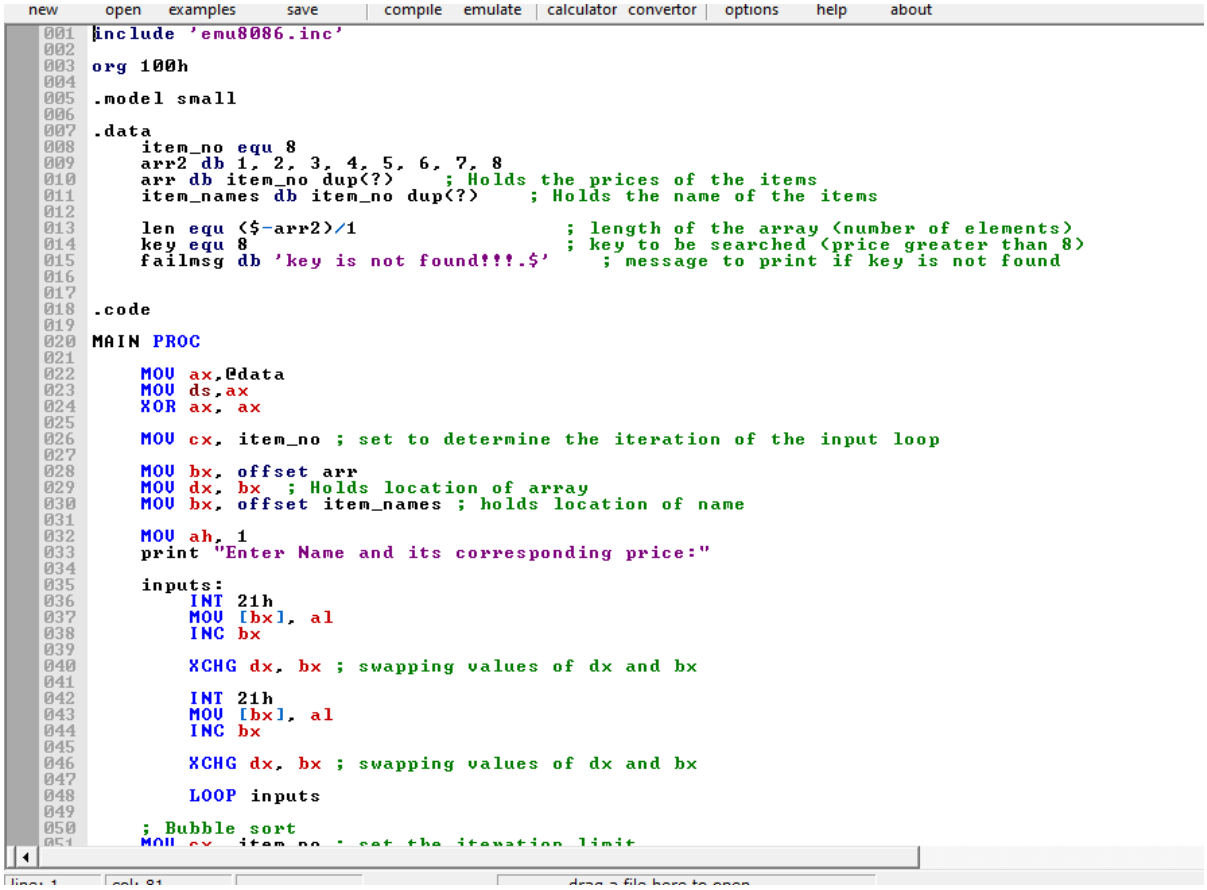
Binary Search Algorithm:

- The binary search algorithm is used to efficiently search for the target number within the sorted array.
- It works by repeatedly dividing the search interval in half until the target number is found or the interval is empty.
- At each step, the algorithm compares the target number with the middle element of the array.
- If the target number matches the middle element, the search is successful. If not, the algorithm narrows down the search interval and repeats the process.
- This process continues until the target number is found or the entire array is searched.

Display Section:

- If the target number is found, the program displays a message indicating success (`FOUND_MSG`).
- If the target number is not found, it displays a message indicating failure (`NOT_FOUND_MSG`).

Full code implementing input processing ,sorting ,binary search algorithm :

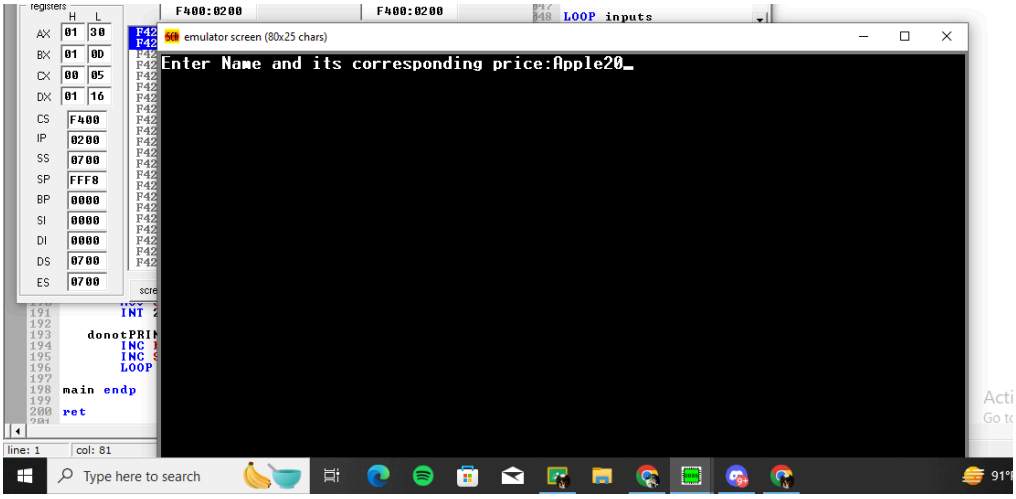



```
include 'emu8086.inc'

org 100h

.model small

.data
    item_no equ 8
    arr2 db 1, 2, 3, 4, 5, 6, 7, 8
    arr db item_no dup(?)    ; Holds the prices of the items
    item_names db item_no dup(?)    ; Holds the name of the items

    len equ ($-arr2)/1                  ; length of the array (number of elements)
    key equ 8                           ; key to be searched (price greater than 8)
    failmsg db 'key is not found!!!.$'    ; message to print if key is not found


.code

MAIN PROC

    MOV ax,@data
    MOV ds,ax
    XOR ax, ax

    MOV cx, item_no ; set to determine the iteration of the input loop

    MOV bx, offset arr
    MOV dx, bx  ; Holds location of array
    MOV bx, offset item_names ; holds location of name

    MOV ah, 1
    print "Enter Name and its corresponding price:"

    inputs:
        INT 21h
        MOV [bx], al
        INC bx

        XCHG dx, bx ; swapping values of dx and bx

        INT 21h
        MOV [bx], al
        INC bx

        XCHG dx, bx ; swapping values of dx and bx

        LOOP inputs

    ; Bubble sort
    MOV cx, item_no ; set the iteration limit
```

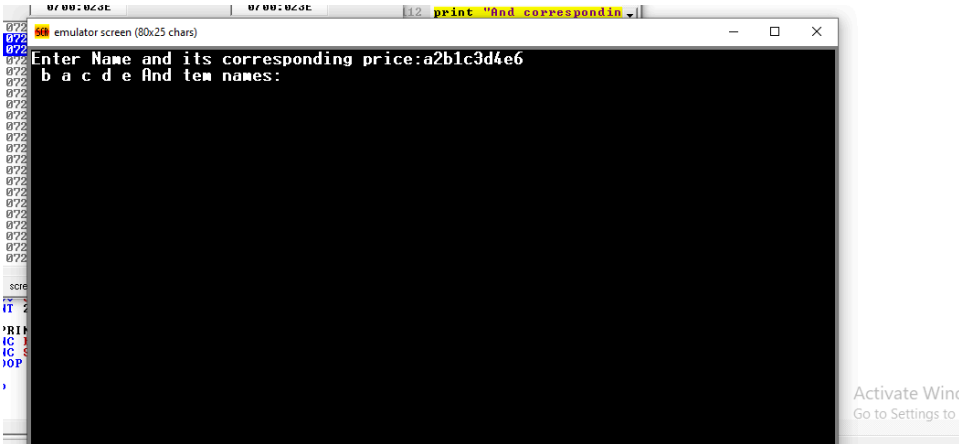
```
049
050     ; Bubble sort
051     MOV cx, item_no ; set the iteration limit
052     DEC cx
053
054 OuterLoop:       ; iterate over each element in arr and comparing
055     MOV bx, cx    ; with all the elements next to it
056     MOV si, 0     ; si is used for indexing the array
057
058 CompLoop:        ; here we compare each consequent elements and sort them if needed
059     MOV al,arr[si]
060     MOV dl,arr[si+1]
061     CMP al,dl
062
063     JC noSwap       ; if the comparator returns none we jump to noSwap
064
065     ; otherwise swapping is done here
066     ; here both the element name and their prices are being swapped
067     MOV arr[si], dl
068     MOV arr[si+1], al
069
070     MOV al,item_names[si]
071     MOV dl,item_names[si+1]
072
073     MOV item_names[si], dl
074     MOV item_names[si+1], al
075
076 noSwap:
077     INC si
078     DEC bx
079     JNZ CompLoop
080
081 LOOP OuterLoop
082
083     ; New line
084     MOV ah,2
085     MOV dl,10
086     INT 21h
087
088     MOV dl, 13
089     INT 21h
090
091     ; Printing sorted item names and corresponding prices
092     print "After Sorting Item names: "
093
094     MOV cx, item_no
095     MOV bx, offset item_names
096
097 Outputs:
098     MOV dl,[bx]
099     MOV ah, 2
```

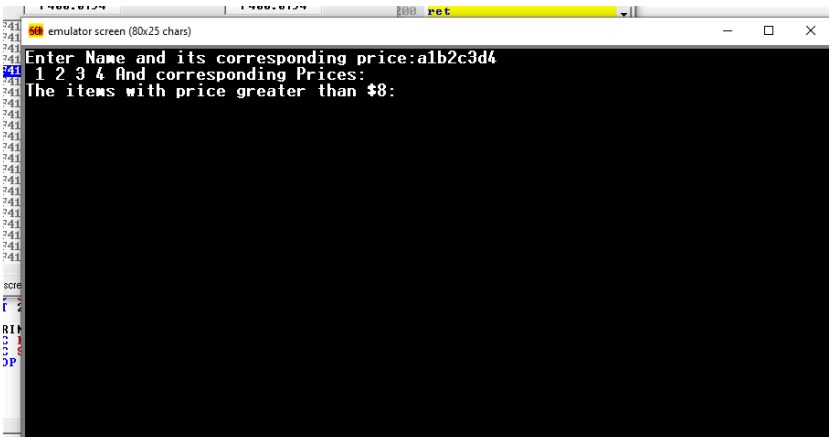
```
097 Outputs:
098     MOV dl,[bx]
099     MOV ah, 2
100     INT 21h
101
102     MOV dl,32
103     MOV ah, 2
104     INT 21h
105
106     INC bx
107     LOOP Outputs
108
109     MOV cx, item_no
110     MOV bx, offset arr
111
112     print "And corresponding Prices: "
113
114 Outputs_2:
115     MOV dl,[bx]
116     MOV ah, 2
117     INT 21h
118
119     MOV dl,32
120     MOV ah, 2
121     INT 21h
122
123     INC bx
124     LOOP Outputs_2
125
126     ; New line
127     MOV ah,2
128     MOV dl,10
129     INT 21h
130
131     MOV dl, 13
132     INT 21h
133
134     ; Binary Search
135     XOR AX, AX   ; Clearing all the register for scary scary
136     XOR BX, BX
137     XOR DX, DX
138     XOR CX, CX
139     XOR SI, SI
140
141     mov bx, 00           ; bx = 0, lower bound
142     mov dx, len          ; dx = len, upper bound
143     mov cx, key          ; cx = key to be searched
144
145 again:
146     cmp bx, dx        ; compare bx with dx
147     ja fail           ; if bx > ax i.e. lower bound > upper bound, jump to
```

```
151     mov si, ax
152     cmp cl, arr2[si]
153     jae big           ; if cl >= mid element, jump to label big else continue
154     dec ax
155     mov dx, ax
156     jmp again
157
158 big:
159     je success        ; if mid element = cl, jump to success else continue
160     inc ax
161     mov bx, ax
162     jmp again
163
164 success:
165     print "The items with price greater than $8: "
166     jmp disp              ; Jump to display routine
167
168 fail:
169     lea dx, failmsg
170
171 disp:
172     MOV cx, item_no
173     MOV bx, offset arr
174     MOV SI, 0
175
176     Outputs_3:
177         MOV al, [bx]
178         SUB al, 30h
179
180         CMP al, key
181
182         JLE donotPRINT  ; if the value is not greater than the key
183                         ; we jump to donotPRINT i.e. we do not print the item name
184
185         MOV dl, item_names[si]
186         MOV ah, 09h
187         INT 21h
188
189         MOV dl, ',', 32
190         MOV ah, 09h
191         INT 21h
192
193     donotPRINT:
194         INC bx
195         INC SI
196         LOOP Outputs_3
197
198 main endp
199
200 ret
```



In this manner the input will be given .for simplicity purpose this name a,b,c,d and corresponding are given .





1. It prompts the user to input item names and their corresponding prices, storing the information in arrays.

2. Utilizes the bubble sort algorithm to arrange the prices in ascending order and displays the complete list of items with their names and sorted prices.

3. Implements the binary search algorithm to find all the items with prices greater than $8 and displays their names.

The code utilizes various instructions like `MOV`, `XCHG`, `INC`, `DEC`, `CMP`, `JE`, `JMP`, `LOOP`, `INT`, and `RET` to perform operations on registers and memory, enabling input, sorting, searching, and output functionalities. The binary search algorithm efficiently locates items with prices greater than $8 within the sorted array.

Questions:

1)Bubble sort Algorithm is used in this case because of simplicity ,low memory use also efficient for small dataset.

- Bubble sort requires minimal additional memory beyond the original array, making it suitable for environments with limited memory resources like the 8086.
- The code provided for bubble sort is relatively simple and easy to understand, which is beneficial for development and maintenance.

2)

Function	Description	Effect
MOV ax,@data	Move the segment address of the data segment into the AX register.	Register operation: Move
MOV ds,ax	Move the value in the AX register to the data segment register.	Register operation: Move

MOV cx, item_no	Move the value of item_no (8) into the CX register.	Register operation: Move
MOV bx, offset arr	Move the offset address of the arr array into the BX register.	Register operation: Move
MOV dx, bx	Move the value in the BX register to the DX register.	Register operation: Move
MOV bx, offset item_names	Move the offset address of the item_names array into the BX register.	Register operation: Move
MOV ah, 1	Move the value 1 into the AH register.	Register operation: Move
INT 21h	Software interrupt for input.	Input/output operation
MOV [bx], al	Move the value in the AL register to the memory location pointed to by BX.	Memory operation: Move
XCHG dx, bx	Exchange the values in the DX and BX registers.	Register operation: Exchange
MOV ah, 2	Move the value 2 into the AH register.	Register operation: Move
LOOP inputs	Loop while CX is not zero, decrementing CX.	Control flow operation: Loop
MOV bx, cx	Move the value in the CX register to the BX register.	Register operation: Move
MOV si, 0	Move the value 0 into the SI register.	Register operation: Move
MOV al,arr[si]	Move the value from the memory	Memory operation: Move

	location pointed to by SI to the AL register.	
MOV dl,arr[si+1]	Move the value from the memory location calculated by adding 1 to the value of SI to the DL register.	Memory operation: Move
CMP al,dl	Compare the values in the AL and DL registers.	Arithmetic/logical operation: Compare
MOV arr[si], dl	Move the value in the DL register to the memory location pointed to by SI.	Memory operation: Move
MOV arr[si+1], al	Move the value in the AL register to the memory location calculated by adding 1 to the value of SI.	Memory operation: Move
MOV al,item_names[si]	Move the value from the memory location pointed to by SI to the AL register.	Memory operation: Move
MOV dl,item_names[si+1]	Move the value from the memory location calculated by adding 1 to the value of SI to the DL register.	Memory operation: Move
MOV item_names[si], dl	Move the value in the DL register to the memory location pointed to by SI.	Memory operation: Move
MOV item_names[si+1], al	Move the value in the AL register to the memory location calculated by adding 1 to the value of SI.	Memory operation: Move
DEC bx	Decrement the value in the BX register by 1.	Arithmetic/logical operation: Decrement

JNZ CompLoop	Jump if Not Zero to CompLoop.	Control flow operation: Jump
MOV dl,10	Move the ASCII value for newline (LF) into the DL register.	Register operation: Move
MOV dl, 13	Move the ASCII value for carriage return (CR) into the DL register.	Register operation: Move
MOV dl,[bx]	Move the value from the memory location pointed to by BX to the DL register.	Memory operation: Move
MOV dl,32	Move the ASCII value for space (' ') into the DL register.	Register operation: Move
INC bx	Increment the value in the BX register by 1.	Arithmetic/logical operation: Increment
MOV al, [bx]	Move the value from the memory location pointed to by BX to the AL register.	Memory operation: Move
SUB al, 30h	Subtract 30h from the value in the AL register.	Arithmetic/logical operation: Subtract
CMP al, key	Compare the value in the AL register with the key.	Arithmetic/logical operation: Compare
JLE donotPRINT	Jump if Less Than or Equal to donotPRINT.	Control flow operation: Jump
MOV dl, item_names[si]	Move the value from the memory location pointed to by SI to the DL register.	Memory operation: Move
MOV dl, ',', 32	Move the ASCII value for comma (',') and space (' ') into the DL register.	Register operation: Mov

MOV al, [bx]	Move the value from the memory location pointed to by BX to the AL register.	Memory operation: Move
INC bx	Increment the value in the BX register by 1.	Arithmetic/logical operation: Increment
INC SI	Increment the value in the SI register by 1.	Arithmetic/logical operation: Increment
LOOP Outputs_3	Loop while CX is not zero, decrementing CX.	Control flow operation: Loop

3) I have found the items that are greater than $9in this case.It is not that efficient handling two decimal digits .The binary search algorithm requires the array to be sorted beforehand, and it is designed to find a single target value efficiently in a sorted array. Regarding the architecture of the 8086, it doesn't have a significant impact on the efficiency of the binary search algorithm itself.
The efficiency of binary search mainly depends on the size of the dataset and whether it is sorted or not. It repeatedly divides the search interval in half until the target value is found or the interval is empty. Once the target value is found, it proceeds to print the corresponding item names.

Discussion:

From this assignment we got to learn a lot of things ,different types of sorting algorithm ,different types of searching algorithm.how to take input from user and store in array .perform operations on the basis of that and so on.We learned to interact with users using interrupt-driven input mechanisms, implemented the bubble sort algorithm for data organization, and utilized binary search to locate specific items within a sorted dataset.

My focus was on optimizing program efficiency by minimizing the number of operations and memory accesses, which is crucial when working within the constraints of the 8086 architecture.