Problem Statement

The problem concerns managing a list of items with their names on an 8086 microprocessor. Given that this task dates back to the 1980s, when spreadsheet software was not widely available and computers were rare, using the 8086 microprocessors proved to be a convenient solution for handling data of purchased items along with their prices. Our task primarily involves storing, sorting, and searching through the input item data.

**Demands:**

* Storing all the Item names with their corresponding prices sequentially.
* Sorting the items based on their prices in ascending order.
* Printing the sorted Items along with their corresponding prices sequentially.
* Employing a search algorithm to find all the Item names above 20 dollars price

Tasks & Solutions

**Tasks:**

Based on the demands our tasks to be performed are:

1. Taking character by character inputs of item names from user.
2. Taking price inputs from user which needs to converted to decimal.
3. Storing the Item names and Prices properly in 2 memory locations.
4. Employing Bubble sort algorithm to sort the prices in ascending manner.
5. Also sorting the corresponding item names with the prices.
6. Printing all the sorted item names with their prices properly.
7. Employing Binary search to find which is the first price element with value greater than 20 dollars.
8. Printing the names from that element (found in step vii) to last of the items name list.

**Solutions:**

**Taking Name Input:**

First, we will declare an ITEM byte array with an arbitrarily chosen size. We have taken size of 400 elements. Here, we have restricted that maximum size of a item name can be 20. So our starting index of each item name will be 20 indices apart.

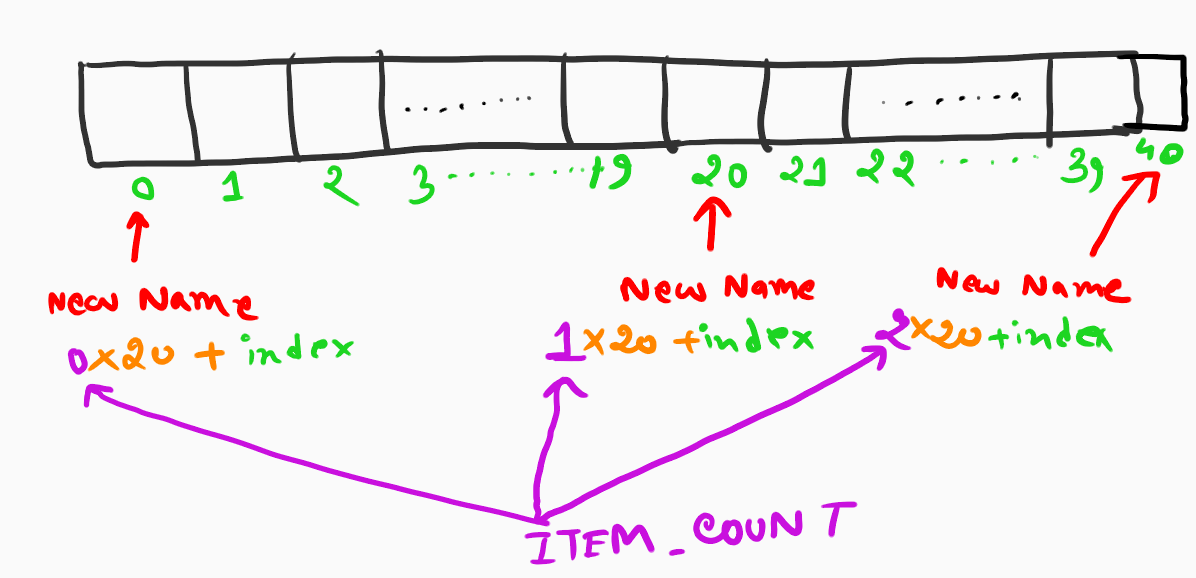


Fig: 01

Here, ITEM\_COUNT is tracking how many items are being added and also being used to offset the starting index for entry of characters for a new name. This entry of characters for one name will keep going until TAB has been pressed. If TAB is pressed, the control will jump to take input of price for that item

**Taking Price Input:**

For taking the price inputs, we have taken a word size array named PRICE. The size of the array has been chosen arbitrarily 20 for storing max 20 items. Each price can be maximum 2559. Due to word type array, each price will be 2 indices apart serially.

Now, for taking input of one price, we have made a user defied function named DECIMAL\_INPUT. This function will initiate the decimal number as N=0. And it will follow the flowchart shown below and finally N will contain our decimal input.

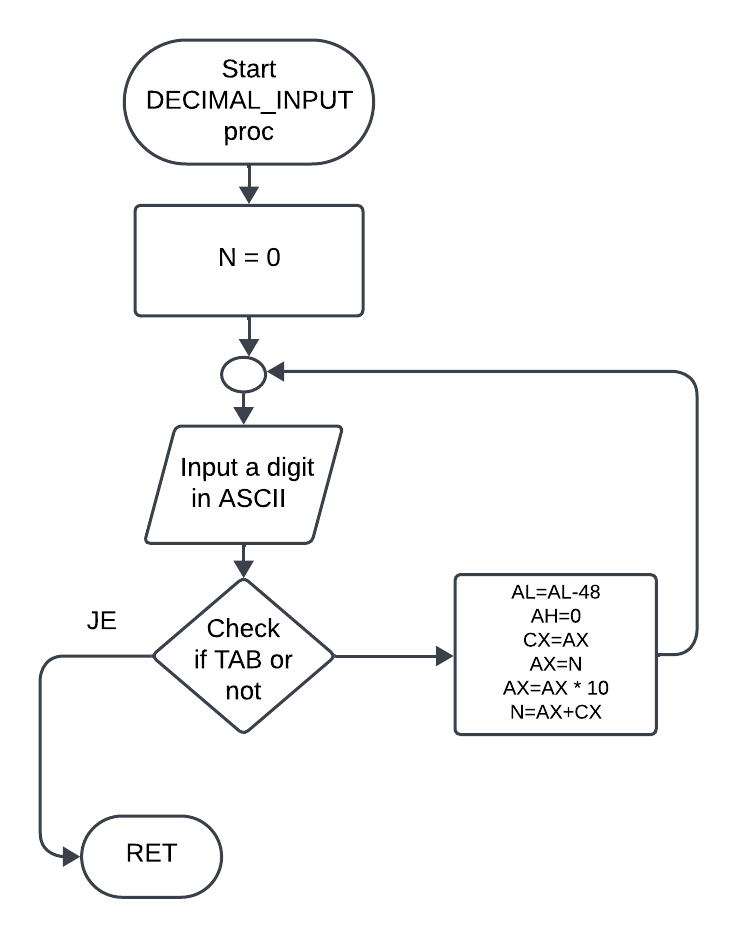


Fig: 02

Now after each Item name and price input, ITEM\_COUNT will be incremented by 1 & the index for tracking PRICE array will be incremented by 2 as this is a word type array. This process will keep going until ENTER has been pressed after pressing the TAB for a price input.

**Employing Bubble Sort algorithm to sort the PRICE array:**

Our bubble sorting algorithm will compare each adjacent price starting from the initial price. This comparison will go till the comparison between the price of item before last item and the price of last item. Due to word size array of prices, we will compare PRICE [index] with PRICE[index+2] and check if swapping is needed or not. If PRICE [index] is larger than PRICE[index+2], we will do an XCHG operation between those 2 elements and increase the index values. Otherwise, we don’t need any swap and we will just increment the index values. A visual representation of Bubble sort is shown below:

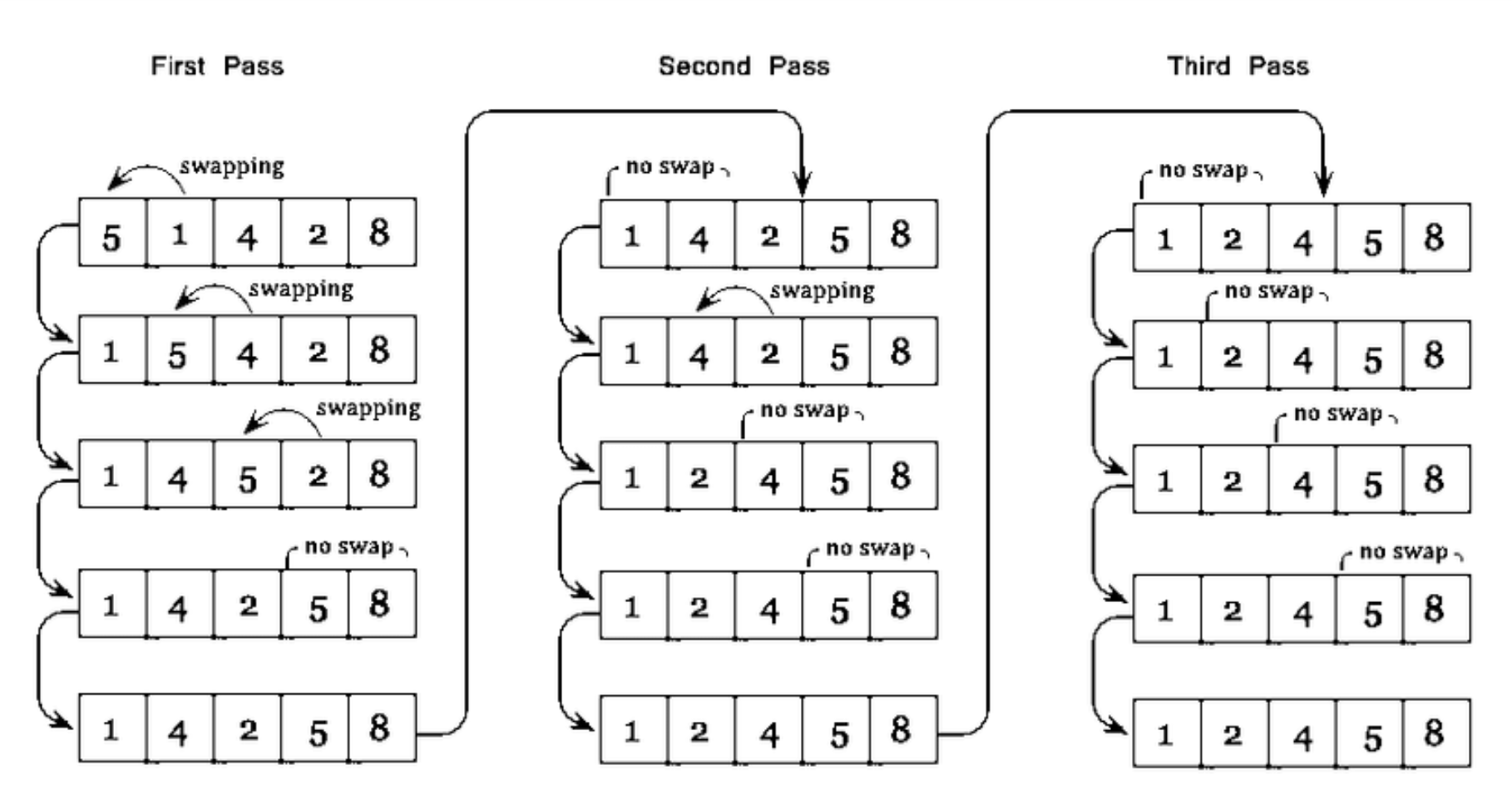


Fig: 03

Here swapping occurs one times less than the ITEM\_COUNT. And each pass here ensures at least one price gets sorted. So, for sorting the entire array, we need to run this whole process for ITEM\_COUNT times.

**Sorting the ITEM array:**

The ITEM array will also get sorted along with the PRICE array. As we have considered length of each item name as 20 characters, we will have to swap 20 characters with each 20 adjacent offset position for completely swapping 2 names. The ITEM\_COUNT\_IDX variable will keep track of which item in serial is getting swapped. Also, for tracking and swapping, we have to find the initial index of both different names from the ITEM array. This can be found from the process shown in the figure below:

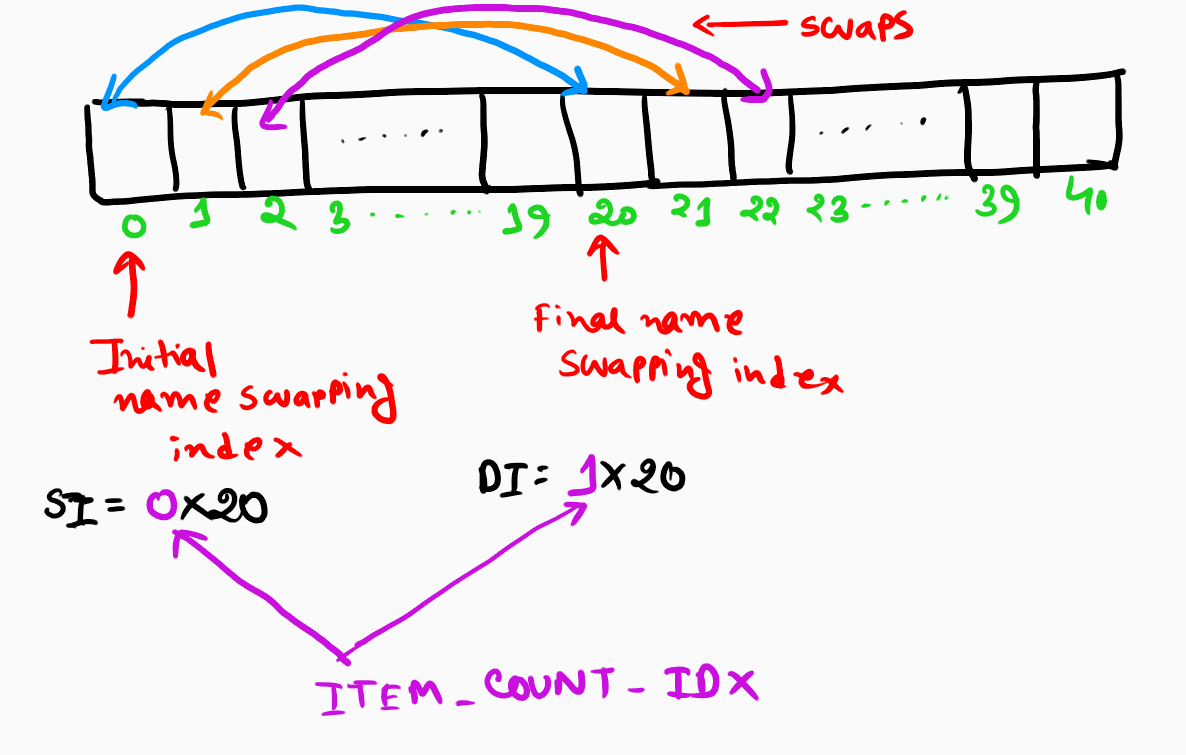


Fig: 04

After Finding 2 adjacent different names starting index, we will store them in SI and DI respectively. Then we will use LOOP keyword run a loop for 20 times to swap the 20 characters serially if needed. After each iteration the SI and DI will get incremented by 1 to keep swapping entire 20 characters.

After swapping the full name the ITEM\_COUNT\_IDX will be incremented by 1 to now swap next 2 names if needed.

If names don’t need any swap then ITEM\_COUNT\_IDX will just increment by 1 without entering the SWAP\_LOOP labeled section.

**Printing the Sorted Item names with their Prices:**

For printing the ITEM names, we will initiate the ITEM\_COUNT\_IDX by 0. We, have taken a variable named NAME\_PRINT\_COUNTER which will track if 20 characters of a name is printed or not. We will use the same initial index finding method as before and put that in ITEM\_COUNT\_IDX. Then we will print each character and increment NAME\_PRINT\_COUNTER by 1 until NAME\_PRINT\_COUNTER gets equal to 19. This variable will be set to 0 every time we start printing a new name.

After printing a name, our program control will jump to PRINT\_PRICE labeled section. For printing the decimal price, we have created a function named DECIMAL\_OUTPUT. Our decimal number is stored in PRICE[SI]. This will be first moved to AX. The flow chart for DECIMAL\_OUTPUT function is shown below:

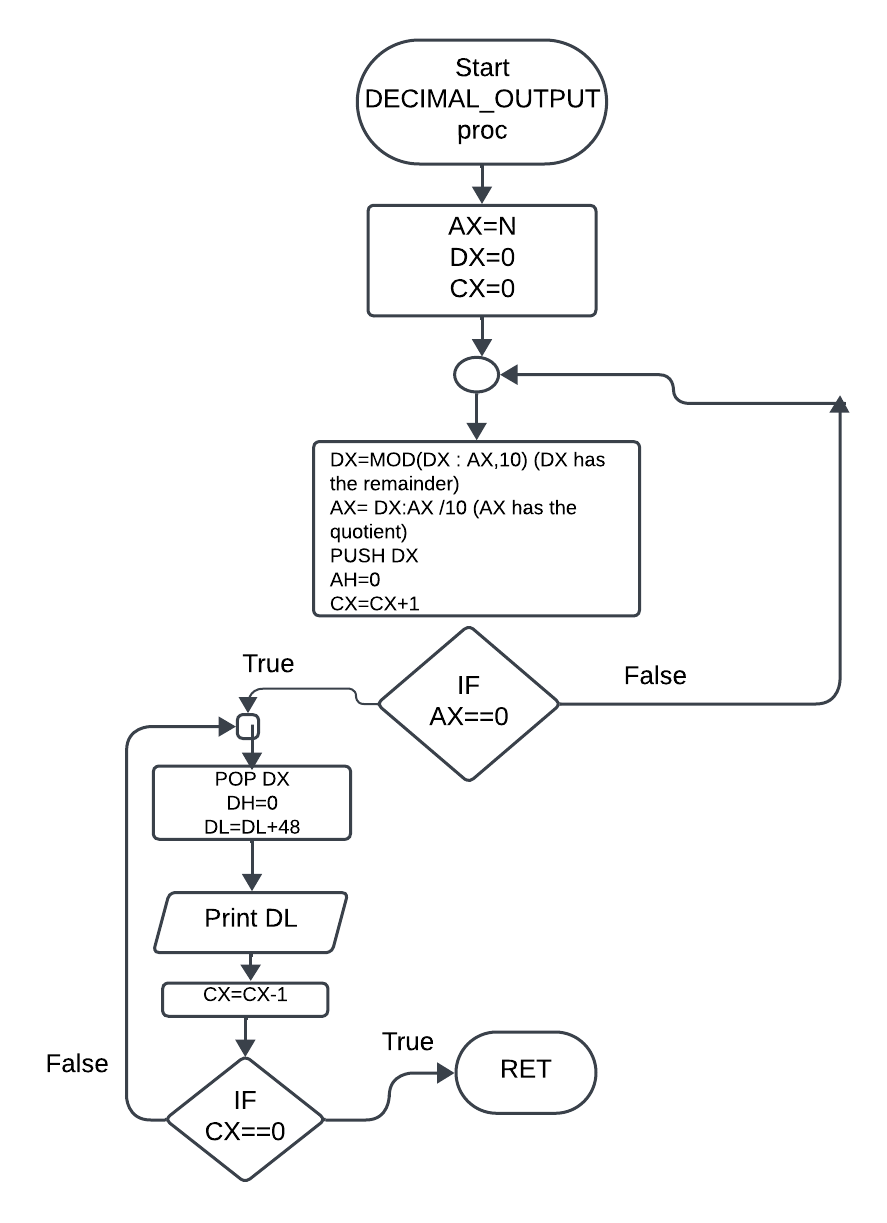


Fig: 05

After printing this price, our program control will again go to label PRINTING, and this will keep going on until ITEM\_COUNT\_IDX exceeds total ITEM\_COUNT.

**Employing Binary Search to Find desired key index:**

For Binary searching, we will first set our previously used ITEM\_COUNT\_IDX and PRICE\_IDX to 0.

First, we will divide the ITEM\_COUNT by 2 to find the middle index from the quotient. As our price array is word sized array, we will multiply by 2 to get the middle index for PRICE array. Now, we will check if this PRICE [middle\_index] value is larger, smaller or equal to 20.

If this is larger than 20 than we will do a backward traverse and where we can find price smaller or equal to 20. Then our desired starting printing index (ITEM\_COUNT\_IDX) will be this middle index+2.

If this is smaller than 20 then we will do a forward traverse and find where we first find items with price higher than 20. That index will be our desired starting printing index (ITEM\_COUNT\_IDX)

**Printing the Items above 20 dollars:**

After finding the desired ITEM\_COUNT\_IDX, we will use this to offset our starting index for printing different names as previously shown. From this offset address it will print next 20 characters using LOOP keyword for NAME\_PRINT2 label.

After printing 20 characters ITEM\_COUNT\_IDX will be incremented by 1 and this process will keep going until ITEM\_COUNT\_IDX gets equal to ITEM\_COUNT.