

**EE213 Computer Organization and Assembly Language Semester Project Report – FALL 2018**

**PROJECT TITLE:**

**“SUPERMARKET MANAGEMENT SYSTEM”**

**PROJECT GROUP\_MEMBERS:**

1. MURAD POPATTIA (17K-3722)
2. SIDDAHANT KUMAR (17K-2372)
3. ALI AKBAR (17K-3871)

**GROUP\_SECTION:** B

**GROUP\_ID:** B08

**COURSE TEACHER:** SIR M.DANISH KHAN

TABLE OF CONTENTS:

1. [ABSTRACT: 3](#_Toc532341410)
2. [PROJECT LAYOUT AND FUNCTIONING: 4](#_Toc532341411)
3. [BRIEF EXPLAINATION OF SOURCE CODE: 5](#_Toc532341413)
4. [SOME LIMITATIONS OF THE CODE: 10](#_Toc532341414)

# ABSTRACT:

Super market, nowadays, are a necessity of a common man leading to the fact that their usage is increasing everyday and hence owners of the super market need some sort of data base in order to have a track regarding the products they have or the items diminished in their overall stock. Furthermore, they also need to keep a track of the everyday sales that the super market accomplishes in order to ensure the smooth economical working of the super market.

The aim of this project was to simulate the working of a super management system, that would cater the need of both customers and the administrator who controls the system and to deliver all statistics regarding sales of the super market to the administration and hence providing a solution to the above mentioned problem. Although this project cannot stand toe to toe with the upcoming technical management systems but rather it represents a model of the base for a basic management system provided the fact that it is made using assembly language. However, maximum effort has been made in order to make the interface as user friendly as possible.

This report will cover topics such as explaining the working of the management system using flow charts and pictures, underlining of some technical features provided and explain how the program code accomplishes each feature, limitations of the project.

# PROJECT LAYOUT AND FUNCTIONING:

## FLOWCHART REPRESENTATION:

D:\Semester 3\COAL\Coal project\Untitled Diagram.png

# BRIEF EXPLAINATION OF SOURCE CODE:

The mechanisms mentioned in the flow chart the features that the program has to offer. Now going on with how the source code covers the functioning of each feature:

**ENTRY PHASE:**

The entry phase of the program requires the user to enter the identity. Considering the user as an admin, the system demands a password so as to set a security line between authorized and non-authorized users. The program does this by comparing the user entered string with a previously saved password string inside .data segment.

mwrite "Enter password: "

mreadstring pass\_check

invoke str\_compare,addr password, addr pass\_check

jne wrong

Else if a customer enters the program allows them to pass through.

**ADMIN PHASE:**

Being an admin, the interface provides the authorized users to do the following tasks:

* **Add product:**

Obviously, an administrator can add items that are new arrivals for his stock. The program accomplishes this by maintaining a product structure.

product STRUCT

id byte "Product id: ",3 DUP(?),0

category byte "Product category: ",10 DUP (?),0

stock byte "Stock: ",5 DUP(?),0

price byte "Price: ",5 DUP(?),0

product ENDS

**STEPS FOLLOWED BY THE CODE:**

1. Take all information regarding product as input.
2. Save product data into a buffer (buffer byte 5000 DUP(?),0).
3. Copy buffer to file “Product Information.txt” from where the admin can analyze all the added data.

* **Delete product:**

The administrator also has the right to delete a product from the database.

**STEPS FOLLOWED BY THE CODE:**

1. Maintain another buffer namely buffer1
2. Take the id to delete as user input, inside a string
3. Read all the product data from file to the buffer
4. Fetch product id from the buffer and compare it to the delete\_id
5. If a match is found, then don’t copy that product’s data inside the buffer1.
6. Else copy all the data of that specific product inside the buffer1.
7. Repeat step 4 until the end of file reached.

* **Update Product:**

The administrator also has the ability to alter the stock or price or the name of the product entered. These are some of the essentials features that the system should have for a good management system.

**STEPS FOLLOWED BY THE CODE:**

1. Take product id to be updated as input.
2. Read all the product data from file to the buffer
3. Fetch product id from the buffer and compare it to the delete\_id
4. If a match is found, generate Msgboxes asking for updating of the fields.
5. Alter the data into the buffer for the respective fields the admin desires to be updated.
6. Repeat step 4 until the end of file reached.

* **Get product List:**

The administrator lastly, can also fetch all the lists the super market has. This list will be the recently updated list after all deletions, updating etc.

**STEPS FOLLOWED BY THE CODE:**

1. Read all the product data from file to the buffer and display it

**CUSTOMER PHASE:**

The interface offers many services for the customer which ranges from ordering items to peeking over to the order. The interface also has an integrated receipt printing accomplished after the checkout. Although this is a management system but we need a simulation of customers ordering in order to show how the sales are maintained. Customers can:

* **Order items:**

Customers have the option to order items by selecting their respective product\_id present in the menu or list given to them. In real time this can be accomplished by the use of barcode scanners.

**STEPS FOLLOWED BY THE CODE:**

1. Read all products data into a buffer
2. Take a product id from user and search it in the buffer
3. If found then check whether the user entered quantity is sufficiently available in the stock
4. If the amount is zero return “Out of stock”
5. Else if the amount is lesser ask the user to enter a value lesser or equal to the amount available
6. Else if sufficient quantity available then place the order
7. The total = quant \* price is stored after order placement
8. Add the order to a receipt buffer that we will use for printing the receipt when checking out

* **Cancel order:**

If the customer doesn’t feel the order was placed accurately, they may cancel their order.

**STEPS FOLLOWED BY THE CODE:**

1. Reset the cart variable maintained to zero
2. Clear the receipt buffer
3. Reset the total to 0

* **Look at order:**

The customer must have the facility to peek over their order in order to ensure the correctness of their speech.

**STEPS FOLLOWED BY THE CODE:**

1. The cart variable will be displayed displaying the number of items ordered
2. This will display the quantity, name, price, id of the product the user ordered

* **Get product List/menu:**

The customer also has the right to see what is available for ordering

**STEPS FOLLOWED BY THE CODE:**

1. Read all the product data from file to the buffer and display it

* **Checkout:**

This is the essential phase for the customer. Checkout means that the customer’s order is finalized and will not be cancelled hence overwriting of the stock process will carry out after the checkout phase. The checkout phase consists of following parts:

1. Generating a random sales\_id denoting the specific customer
2. Overwriting from the stock and updating new values
3. Marking the end of customer phase

**STEPS FOLLOWED BY THE CODE:**

1. A sales structure is maintained in the code along with the sales\_buffer which ensures the proper recording and maintaining of the sales data

sales STRUCT

id byte "Sales id: ",5 DUP(?),0

sellprod\_id byte "Product id: ",3 DUP(?),0

sale\_price byte "Price: ",5 DUP(?),0

sale\_quantity byte "Quantity: ",5 DUP(?),0

total byte "Transcation amount: ",7 DUP(?),0

sales ENDS

1. A systime struct is also used in order to record the time and date at which the sales occurred. This date is written with every new generated sales\_id inside the sales\_buffer

systime SYSTEMTIME <>

1. The sales id along with the ordered products present in the receipt buffer are processed and written into the sales buffer along with the transaction total of the order
2. The quantities of products ordered by the customer are then altered and new values for the stock are updated inside the buffer. This new updated product buffer is then written back to the “Product Information.txt” file stored.
3. The sales buffer is then written to a file “Sales Information.txt”.
4. After the records have been maintained, the respective receipt is printed and the phase ends.

**UTILITY FUNCTIONS:**

Some utility functions listed below have been used too in order to assist the smooth functioning of the code

* **Convert\_to\_integer proc:**

This converts a number stored in string format inside str1 to a number which is then stored in EAX

* **Convert\_to\_string proc:**

This converts a number stored in EAX to string which is stored inside str1 present in the memory

* **Read/Write buffers proc:**

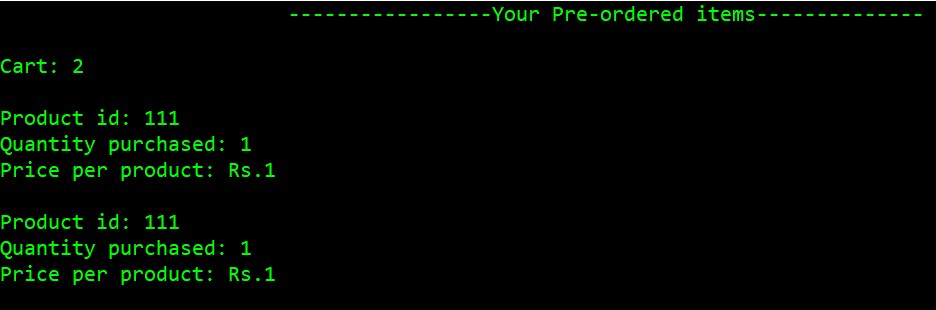
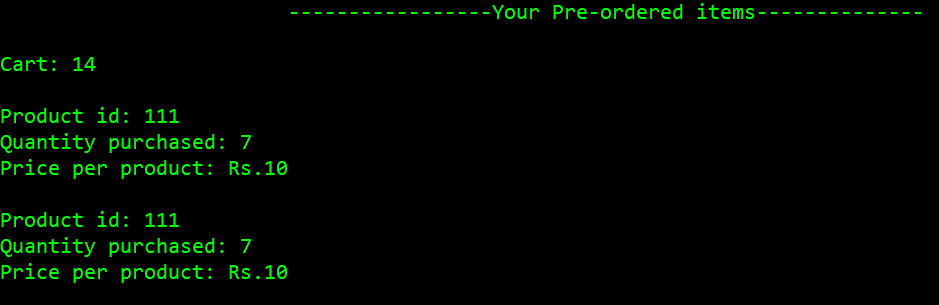
At multiple points these pieces of codes have been used due to their repetitiveness and requirement and hence been made into a utility function

* **Generate\_sales\_id proc:**

This generates a random sales\_id and stores it inside sales.id field

# SOME LIMITATIONS OF THE CODE:

Although effort has been made to ensure its effectiveness, there still rests some imperfections which this project fails to cater:

1. To add new fields of information for the products
2. The admin is limited to enter the product id to 3 digits specific until altered by the programmer which is not a real time occurring and hence is limited
3. The code, due to its massive functional calling is quite slow and not made for speedy systems.
4. Ordering an item multiple times will show the item multiple times instead of overwriting its ordered quantity:
5. Presence of ‘!’ at the end of file may seem quite annoying for some users although this is very essential for this program to denote the end of a file.
6. Check\_in\_stock functionality only checks the fact that the quantity entered is lesser than that in stock. However a customer may do something like this:   
     
     
     
     
     
     
     
     
     
   ignoring the fact that the stock contains only 10 products of prod\_id:111. Now during checkout the stock will be updated to a negative value giving an error hence this is also an uncorrected limitation of the code.