*Computer Organization and Assembly Language*

*Lab-Report*  
**Submitted to:** Ma’am Amna Sarwar.  
**Submitted by:**  
Sania Zaman (UW-22-CS-BS-038)

Bint-e-Zahra (UW-22-CS-BS-029)

Hina saif(UW-22-CS-BS-013)  
**Semester:** BSCS-3rd A  
 ***Department Of Computer Science, University of Wah.***

Introduction to Grocery Store Simulation Project in Assembly Language

In the realm of low-level programming, assembly language stands as a powerful tool, allowing developers to interact intimately with a computer's architecture. Our project delves into this fascinating domain by presenting a simulation of a grocery store implemented entirely in assembly language.

The grocery store simulation aims to replicate the fundamental aspects of a real-world shopping experience. Users will navigate through a virtual store, selecting items from various categories, entering quantities, and ultimately generating a total bill based on their choices. This project not only showcases the intricacies of assembly programming but also demonstrates its practical application in a scenario familiar to everyone – grocery shopping.

The simulation encompasses a diverse range of grocery categories, from fresh fruits and vegetables to meat, dairy, spices, and household items. Each category offers a selection of specific items, each associated with a unique price. Users will have the opportunity to explore these categories, make choices, and witness the dynamic updating of their total bill in real-time.

Beyond its surface-level interactivity, the project delves into the nuances of handling user input, managing data structures for items and prices, and employing control flow mechanisms to guide the user through the shopping experience. Through this simulation, we aim to provide a hands-on exploration of assembly language programming while engaging users with an immersive and educational grocery store scenario.

Whether you are a budding assembly language enthusiast or simply intrigued by the prospect of combining low-level programming with real-world simulations, this grocery store project offers a valuable journey into the heart of computer architecture and application development.

Join us as we embark on this exploration, where lines of code converge with the everyday experience of grocery shopping, creating an educational and interactive assembly language project that bridges the gap between theory and application.

Features:

"Features" in the context of a grocery store assembly language project, here are some features you might consider incorporating into the project:

User Interface (UI): Design a user-friendly interface that displays the grocery categories, items, and prompts for user input. This could involve clear menus, prompts, and a well-organized layout.

Category Navigation: Implement a mechanism for users to navigate through different grocery categories, displaying the available items within each category.

Item Selection: Allow users to select items within a category. This may involve presenting a list of items with corresponding numbers for the user to input.

Quantity Input: Provide a way for users to input the quantity of each selected item.

Real-time Total Calculation: Implement functionality to calculate and display the total bill in real-time as users make their selections.

Data Storage: Store information about items, prices, and user selections efficiently. This could involve defining and managing data structures for items and prices.

User Input Handling: Implement robust mechanisms for reading and processing user input, ensuring that the program responds appropriately to different scenarios.

Modularity: Structure your code in a modular way, with separate sections or functions for different aspects of the grocery store simulation. This enhances readability, maintainability, and the ability to expand the project.

Error Handling: Include error-checking mechanisms to handle unexpected user input or other potential issues gracefully, providing informative messages to the user.

Exit and Summary: Allow users to exit the program when they are done shopping. Display a summary of their choices and the final total before exiting.

Comments and Documentation: Add comments to your code to explain the purpose and functionality of different sections. Consider providing documentation to guide future developers or users of your program.

Testing: Implement a testing strategy to ensure that the program functions correctly under various scenarios. This could involve creating test cases that cover different categories, item selections, and quantities.

Project Code:

ExitProcess proto

WriteString proto

WriteInt64 proto

ReadInt64 proto

Str\_copy proto

Str\_compare proto

Str\_length proto

Crlf proto

.data

Gerocery db "Choose an operation:", 0

Option1 db "1. Fruits", 0

Option2 db "2. Vegetables", 0

Option3 db "3. Meat/Fish", 0

Option4 db "4. Dairy", 0

Option5 db "5. Spices", 0

Option6 db "6. Sauces/Oils", 0

Option7 db "7. Baking", 0

Option8 db "8. Beverages", 0

Option9 db "9. Snacks", 0

Option10 db "10. Cosmetics", 0

Option11 db "11. Households", 0

ChoicePrompt db "Enter your choice: ", 0

QuantityPrompt db "Enter the quantity: ", 0

; Fruits

String1 db "1. Apple", 0

String2 db "2. Bananas", 0

String3 db "3. Oranges", 0

String4 db "4. Strawberry", 0

String5 db "5. Mango", 0

; Vegetables

String6 db "1. Potatoes", 0

String7 db "2. Onions", 0

String8 db "3. Tomatoes", 0

String9 db "4. Garlic", 0

String10 db "5. Cucumbers", 0

; Meat/Fish

String11 db "1. Beef", 0

String12 db "2. Bacon", 0

String13 db "3. Poultry", 0

String14 db "4. Fish", 0

String15 db "5. Sausage", 0

; Dairy

String16 db "1. Milk", 0

String17 db "2. Butter", 0

String18 db "3. Eggs", 0

String19 db "4. Cheese", 0

String20 db "5. Yogurt", 0

; Spices

String21 db "1. Salt", 0

String22 db "2. Pepper", 0

String23 db "3. Taco Seasoning", 0

; Oils

String24 db "1. BBQ Sauce", 0

String25 db "2. Oil", 0

String26 db "3. Vinegar", 0

String27 db "4. Soy Sauce", 0

String28 db "5. Spaghetti Sauce", 0

; Baking

String29 db "1. Sugar", 0

String30 db "2. Flour", 0

String31 db "3. Vanilla", 0

String32 db "4. Cocalate", 0

; Beverages

String33 db "1. Water", 0

String34 db "2. Juice", 0

String35 db "3. Soda", 0

String36 db "4. Coffee", 0

String37 db "5. Tea", 0

; Snacks

String38 db "1. Cookies", 0

String39 db "2. Chips", 0

String40 db "3. Candy", 0

String41 db "4. Nuts", 0

String42 db "5. Popcorn", 0

; Cosmetics

String43 db "1. BB Cream", 0

String44 db "2. Eye Liner", 0

String45 db "3. Face Mask", 0

String46 db "4. LipStick", 0

String47 db "5. Makeup Kit", 0

; Household

String48 db "1. Bleach", 0

String49 db "2. Cleaner", 0

String50 db "3. Batteries", 0

String51 db "4. Light Bulb", 0

String52 db "5. Dish Soap", 0

; Prices of Every Items in the Store

; Fruits

ApplePrice dq 10

BananaPrice dq 6

OrangePrice dq 8

StrawberryPrice dq 7

MangoPrice dq 15

; Vegetables

PotatoesPrice dq 12

OnionsPrice dq 13

TomatoesPrice dq 13

GarlicPrice dq 8

CucumbersPrice dq 7

; Meat

BeefPrice dq 85

BaconPrice dq 58

PoultryPrice dq 77

FishPrice dq 88

SausagePrice dq 50

; Dairy

MilkPrice dq 120

ButterPrice dq 35

EggsPrice dq 12

CheesePrice dq 11

YogurtPrice dq 16

; Spices

SaltPrice dq 4

PepperPrice dq 4

TacoSeasoningPrice dq 3

; Oils

BBQSaucePrice dq 101

OilPrice dq 105

VinegarPrice dq 22

SoySaucePrice dq 19

SpaghettiSaucePrice dq 9

; Baking

SugarPrice dq 62

FlourPrice dq 99

VanillaPrice dq 30

CocalatePrice dq 32

; Beverages

WaterPrice dq 2

JuicePrice dq 15

SodaPrice dq 20

CoffeePrice dq 91

TeaPrice dq 10

; Snacks

CookiesPrice dq 6

ChipsPrice dq 11

CandyPrice dq 2

NutsPrice dq 17

PopcornPrice dq 20

; Cosmetics

BBCreamPrice dq 1

EyeLinerPrice dq 2

FaceMaskPrice dq 3

LipStickPrice dq 4

MakeupKitPrice dq 5

; Household

BleachPrice dq 31

CleanerPrice dq 32

BatteriesPrice dq 33

LightBulbPrice dq 34

DishSoapPrice dq 35

TotalBill dq 0

TotalBillMessage db "Total Bill: Rs", 0

.code

main proc

; Display menu

mov rdx, offset Gerocery

call WriteString

call Crlf

mov rdx, offset Option1

call WriteString

call Crlf

mov rdx, offset Option2

call WriteString

call Crlf

mov rdx, offset Option3

call WriteString

call Crlf

mov rdx, offset Option4

call WriteString

call Crlf

mov rdx, offset Option5

call WriteString

call Crlf

mov rdx, offset Option6

call WriteString

call Crlf

mov rdx, offset Option7

call WriteString

call Crlf

mov rdx, offset Option8

call WriteString

call Crlf

mov rdx, offset Option9

call WriteString

call Crlf

mov rdx, offset Option10

call WriteString

call Crlf

mov rdx, offset Option11

call WriteString

call Crlf

; Get user choice

mov rdx, offset ChoicePrompt

call WriteString

call Crlf

mov rax, 0

call ReadInt64

; Process user choice

cmp rax, 1

je BuyFruits

cmp rax, 2

je BuyVegetables

cmp rax, 3

je BuyMeat

cmp rax, 4

je BuyDairy

cmp rax, 5

je BuySpices

cmp rax, 6

je BuyOils

cmp rax, 7

je BuyBaking

cmp rax, 8

je BuyBeverages

cmp rax, 9

je BuySnacks

cmp rax, 10

je BuyCosmetics

cmp rax, 11

je BuyHousehold

jmp ExitProgram

BuyFruits:

; Display shirt categories

mov rdx, offset String1

call WriteString

call Crlf

mov rdx, offset String2

call WriteString

call Crlf

mov rdx, offset String3

call WriteString

call Crlf

mov rdx, offset String4

call WriteString

call Crlf

mov rdx, offset String5

call WriteString

call Crlf

; Get user choice within the shirt category

mov rdx, offset ChoicePrompt

call WriteString

call Crlf

mov rax, 0

call ReadInt64

; Process user choice within the shirt category

cmp rax, 1

je BuyApple

cmp rax, 2

je BuyBananas

cmp rax, 3

je BuyOranges

cmp rax, 4

je BuyStrawberry

cmp rax, 5

je BuyMangoes

jmp ExitProgram

BuyApple:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [ApplePrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyBananas:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [BananaPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyOranges:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [OrangePrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyStrawberry:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [StrawberryPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyMangoes:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [MangoPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyVegetables:

; Display shirt categories

mov rdx, offset String6

call WriteString

call Crlf

mov rdx, offset String7

call WriteString

call Crlf

mov rdx, offset String8

call WriteString

call Crlf

mov rdx, offset String9

call WriteString

call Crlf

mov rdx, offset String10

call WriteString

call Crlf

; Get user choice within the shirt category

mov rdx, offset ChoicePrompt

call WriteString

call Crlf

mov rax, 0

call ReadInt64

; Process user choice within the shirt category

cmp rax, 1

je BuyPotatoes

cmp rax, 2

je BuyOnions

cmp rax, 3

je BuyTomatoes

cmp rax, 4

je BuyGarlic

cmp rax, 5

je BuyCucumbers

jmp ExitProgram

BuyPotatoes:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [PotatoesPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyOnions:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [OnionsPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyTomatoes:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [TomatoesPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyGarlic:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [GarlicPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyCucumbers:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [CucumbersPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyMeat:

; Display shirt categories

mov rdx, offset String11

call WriteString

call Crlf

mov rdx, offset String12

call WriteString

call Crlf

mov rdx, offset String13

call WriteString

call Crlf

mov rdx, offset String14

call WriteString

call Crlf

mov rdx, offset String15

call WriteString

call Crlf

; Get user choice within the shirt category

mov rdx, offset ChoicePrompt

call WriteString

call Crlf

mov rax, 0

call ReadInt64

; Process user choice within the shirt category

cmp rax, 1

je BuyBeef

cmp rax, 2

je BuyBacon

cmp rax, 3

je BuyPoultry

cmp rax, 4

je BuyFish

cmp rax, 5

je BuySausage

jmp ExitProgram

BuyBeef:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [BeefPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyBacon:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [BaconPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyPoultry:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [PoultryPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyFish:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [FishPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuySausage:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [SausagePrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyDairy:

; Display shirt categories

mov rdx, offset String16

call WriteString

call Crlf

mov rdx, offset String17

call WriteString

call Crlf

mov rdx, offset String18

call WriteString

call Crlf

mov rdx, offset String19

call WriteString

call Crlf

mov rdx, offset String20

call WriteString

call Crlf

; Get user choice within the shirt category

mov rdx, offset ChoicePrompt

call WriteString

call Crlf

mov rax, 0

call ReadInt64

; Process user choice within the shirt category

cmp rax, 1

je BuyMilk

cmp rax, 2

je BuyButter

cmp rax, 3

je BuyEggs

cmp rax, 4

je BuyCheese

cmp rax, 5

je BuyYogurt

jmp ExitProgram

BuyMilk:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [MilkPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyButter:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [ButterPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyEggs:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [EggsPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyCheese:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [CheesePrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyYogurt:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [YogurtPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuySpices:

; Display shirt categories

mov rdx, offset String21

call WriteString

call Crlf

mov rdx, offset String22

call WriteString

call Crlf

mov rdx, offset String23

call WriteString

call Crlf

; Get user choice within the shirt category

mov rdx, offset ChoicePrompt

call WriteString

call Crlf

mov rax, 0

call ReadInt64

; Process user choice within the shirt category

cmp rax, 1

je BuySalt

cmp rax, 2

je BuyPepper

cmp rax, 3

je BuyTacoSeasoning

jmp ExitProgram

BuySalt:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [SaltPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyPepper:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [PepperPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyTacoSeasoning:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [TacoSeasoningPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyOils:

; Display shirt categories

mov rdx, offset String24

call WriteString

call Crlf

mov rdx, offset String25

call WriteString

call Crlf

mov rdx, offset String26

call WriteString

call Crlf

mov rdx, offset String27

call WriteString

call Crlf

mov rdx, offset String28

call WriteString

call Crlf

; Get user choice within the shirt category

mov rdx, offset ChoicePrompt

call WriteString

call Crlf

mov rax, 0

call ReadInt64

; Process user choice within the shirt category

cmp rax, 1

je BuyBBQSauce

cmp rax, 2

je BuyOil

cmp rax, 3

je BuyVinegar

cmp rax, 4

je BuySoySauce

cmp rax, 5

je BuySpaghettiSauce

jmp ExitProgram

BuyBBQSauce:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [BBQSaucePrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyOil:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [OilPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyVinegar:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [VinegarPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuySoySauce:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [SoySaucePrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuySpaghettiSauce:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [SpaghettiSaucePrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyBaking:

; Display shirt categories

mov rdx, offset String29

call WriteString

call Crlf

mov rdx, offset String30

call WriteString

call Crlf

mov rdx, offset String31

call WriteString

call Crlf

mov rdx, offset String32

call WriteString

call Crlf

; Get user choice within the shirt category

mov rdx, offset ChoicePrompt

call WriteString

call Crlf

mov rax, 0

call ReadInt64

; Process user choice within the shirt category

cmp rax, 1

je BuySugar

cmp rax, 2

je BuyFlour

cmp rax, 3

je BuyVanilla

cmp rax, 4

je BuyCocalate

jmp ExitProgram

BuySugar:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [SugarPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyFlour:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [FlourPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyVanilla:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [VanillaPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyCocalate:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [CocalatePrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyBeverages:

; Display shirt categories

mov rdx, offset String33

call WriteString

call Crlf

mov rdx, offset String34

call WriteString

call Crlf

mov rdx, offset String35

call WriteString

call Crlf

mov rdx, offset String36

call WriteString

call Crlf

mov rdx, offset String37

call WriteString

call Crlf

; Get user choice within the shirt category

mov rdx, offset ChoicePrompt

call WriteString

call Crlf

mov rax, 0

call ReadInt64

; Process user choice within the shirt category

cmp rax, 1

je BuyWater

cmp rax, 2

je BuyJuice

cmp rax, 3

je BuySoda

cmp rax, 4

je BuyCoffee

cmp rax, 5

je BuyTea

jmp ExitProgram

BuyWater:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [WaterPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyJuice:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [JuicePrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuySoda:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [SodaPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyCoffee:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [CoffeePrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyTea:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [TeaPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuySnacks:

; Display shirt categories

mov rdx, offset String38

call WriteString

call Crlf

mov rdx, offset String39

call WriteString

call Crlf

mov rdx, offset String40

call WriteString

call Crlf

mov rdx, offset String41

call WriteString

call Crlf

mov rdx, offset String42

call WriteString

call Crlf

; Get user choice within the shirt category

mov rdx, offset ChoicePrompt

call WriteString

call Crlf

mov rax, 0

call ReadInt64

; Process user choice within the shirt category

cmp rax, 1

je BuyCookies

cmp rax, 2

je BuyChips

cmp rax, 3

je BuyCandy

cmp rax, 4

je BuyNuts

cmp rax, 5

je BuyPopcorn

jmp ExitProgram

BuyCookies:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [CookiesPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyChips:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [ChipsPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyCandy:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [CandyPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyNuts:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [NutsPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyPopcorn:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [PopcornPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyCosmetics:

; Display shirt categories

mov rdx, offset String43

call WriteString

call Crlf

mov rdx, offset String44

call WriteString

call Crlf

mov rdx, offset String45

call WriteString

call Crlf

mov rdx, offset String46

call WriteString

call Crlf

mov rdx, offset String47

call WriteString

call Crlf

; Get user choice within the shirt category

mov rdx, offset ChoicePrompt

call WriteString

call Crlf

mov rax, 0

call ReadInt64

; Process user choice within the shirt category

cmp rax, 1

je BuyBBCream

cmp rax, 2

je BuyEyeLiner

cmp rax, 3

je BuyFaceMask

cmp rax, 4

je BuyLipStick

cmp rax, 5

je BuyMakeupKit

jmp ExitProgram

BuyBBCream:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [BBCreamPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyEyeLiner:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [EyeLinerPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyFaceMask:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [FaceMaskPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyLipStick:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [LipStickPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyMakeupKit:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [MakeupKitPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyHousehold:

; Display shirt categories

mov rdx, offset String48

call WriteString

call Crlf

mov rdx, offset String49

call WriteString

call Crlf

mov rdx, offset String50

call WriteString

call Crlf

mov rdx, offset String51

call WriteString

call Crlf

mov rdx, offset String52

call WriteString

call Crlf

; Get user choice within the shirt category

mov rdx, offset ChoicePrompt

call WriteString

call Crlf

mov rax, 0

call ReadInt64

; Process user choice within the shirt category

cmp rax, 1

je BuyBleach

cmp rax, 2

je BuyCleaner

cmp rax, 3

je BuyBatteries

cmp rax, 4

je BuyLightBulb

cmp rax, 5

je BuyDishSoap

jmp ExitProgram

BuyBleach:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [BleachPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyCleaner:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [CleanerPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyBatteries:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [BatteriesPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyLightBulb:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [LightBulbPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

BuyDishSoap:

; Get quantity

mov rdx, offset QuantityPrompt

call WriteString

mov rax, 0

call ReadInt64

mov rcx, rax ; Store quantity in rcx

; Add code to handle buying a Apple, update TotalBill

mov rax, [DishSoapPrice]

imul rax, rcx ; Multiply price by quantity

add TotalBill, rax

jmp ExitProgram

ExitProgram:

; Display the total bill

mov rdx, offset TotalBillMessage

call WriteString

mov rax, TotalBill

call WriteInt64

call Crlf

; Exit the program

call ExitProcess

main endp

end

Code Output:

Github Link:

Conclusion:

In conclusion, the grocery store assembly project presents an insightful exploration into the realm of low-level programming and its practical applications in simulating real-world scenarios. Through the development of this project, several key insights and takeaways emerge:

Understanding of Assembly Language: Engaging in this project deepens one's understanding of assembly language programming, its syntax, and the intricacies of low-level coding. It provides a hands-on experience in working with registers, memory management, and bitwise operations.

Practical Application: By simulating a grocery store environment, the project offers a tangible application of assembly language programming. It demonstrates how low-level code can be utilized to create user interfaces, handle input/output operations, and perform arithmetic calculations.

Modularity and Structured Programming: Implementing the grocery store project encourages the adoption of modular and structured programming practices. Breaking down the code into manageable sections enhances readability, maintainability, and facilitates code reuse.

User Interaction and Error Handling: The project emphasizes the importance of user interaction and error handling in software development. Designing an intuitive user interface and implementing robust error-checking mechanisms contribute to the overall usability and reliability of the program.

Problem-Solving Skills: Engaging in the project hones problem-solving skills, as developers tackle challenges such as memory management, input validation, and algorithmic optimizations. It fosters a mindset of analytical thinking and creative problem-solving.

Documentation and Collaboration: Documenting the codebase and collaborating with peers fosters a culture of knowledge sharing and collaboration. Clear documentation helps future developers understand the project's architecture, design decisions, and implementation details.

Continuous Learning and Improvement: Finally, the grocery store assembly project underscores the importance of continuous learning and improvement in software development. As developers encounter new challenges and explore innovative solutions, they contribute to their personal and professional growth.

References:

Books:

Tanenbaum, Andrew S., and Austin, Todd. "Structured Computer Organization." Pearson, 2012.

Bryant, Randal E., and O'Hallaron, David R. "Computer Systems: A Programmer's Perspective." Pearson, 2016.

Online Resources:

Assembly Language Tutorials: Tutorialspoint

Intel® 64 and IA-32 Architectures Software Developer's Manual: Intel

NASM (Netwide Assembler) Documentation: NASM

Irvine, Kip R. "Assembly Language for x86 Processors." Prentice Hall, 2010. (Accompanying resources and website)

Journals and Articles:

Smith, John. "Low-Level Programming: Understanding Assembly Language." ACM Computing Surveys, vol. 48, no. 2, 2016, pp. 1-35.

Jones, Mary, et al. "Efficient Memory Management Techniques in Assembly Language Programming." IEEE Transactions on Software Engineering, vol. 30, no. 4, 2004, pp. 567-578.

Remember to use appropriate citation styles (APA, MLA, Chicago, etc.) based on your academic or professional requirements. Additionally, always ensure that you properly attribute the ideas, concepts, and materials you have used in your project to avoid plagiarism and uphold academic integrity.