**CSC304 – Microprocessor Architecture and Assembly**

**Project Description**

**VENDING MACHINE IMPLEMENTATION**

**Prepared by:**

**Introduction:**

A vending machine is a machine that gives out different kinds of products when a person inserts a coin into it; therefore it is a coin based vending machine. These machines can be implemented using various methods but in this project it has been implemented using 8051 microcontroller. Vending machines makes it easier for making small purchases, it occupies less space and it doesn’t need any continuous monitoring.The Usage of these machines is increasing day by day. It reduces the human effort and efficiency of the work also increases.

Generally, vending machines become a wide channel with increase in sales and even the

Competition between the manufactures. It has many benefits, first benefit is in terms of setup, it is very easy to setup a vending machine as it takes very little space and it is very compact in size, it is a low cost driven machine which can deliver many products as output.

**Flow Chart:**

Show 4 Items on LCD

With Price

Enter item number

Enter price by Coin

Compare entered coin with item price

Motor start for some time

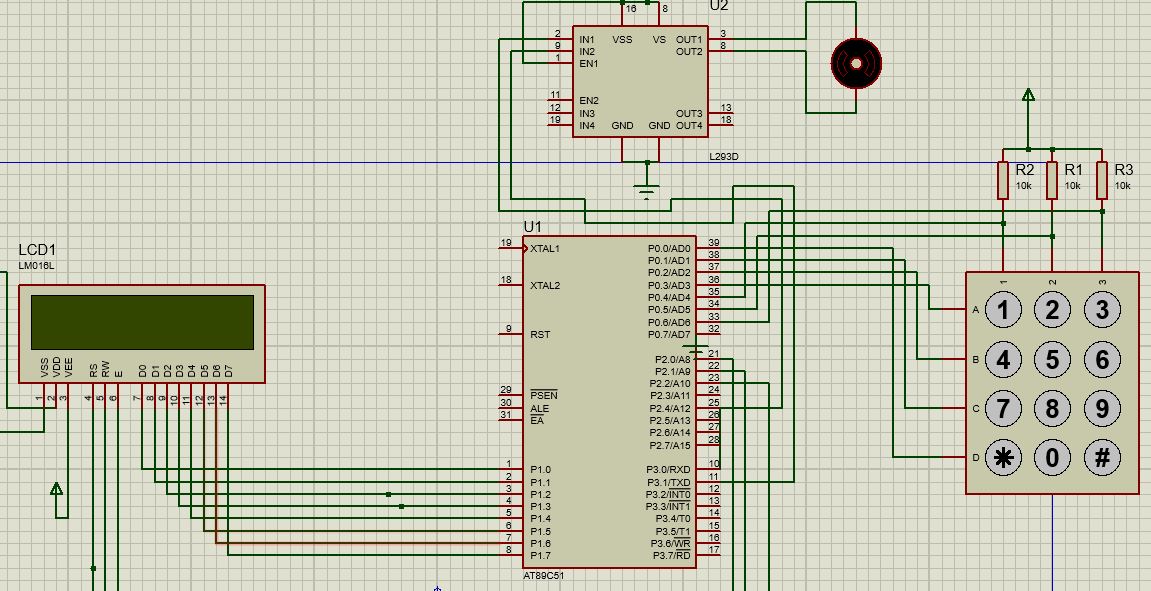
If wr If correct coin entered

If price don’t match

Motor remain off

The flowchart explains the whole working of this machine. In the first step all the items like which may be 4 or 5 or more are displayed on the screen with their fixed prices and serial numbers. If user wants to buy something then he selects the respective items via the keypad and after that the price entering process comes. The user enters the price of respective item via the coin. The machine accepts this coin and compares the price of the entered coin with the actual price of the item. If the price entered is equal to the actual price of the item then the motor runs at full speed for some time which is the indication that the respective item is delivered to the user. If the entered price is not equal to the actual of the item then the motors does not run and LCD moves back to its actual main page which is showing the price of all the items.

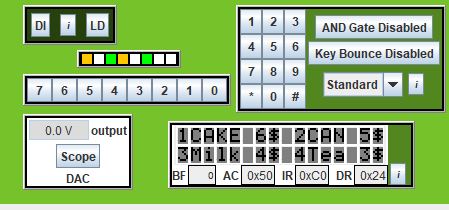
**Circuit Diagram:**



The circuit diagram shows all the components added in this project. The main Component is the 8051 microcontroller which is the heart of the project and it is responsible for all the processing of the code. Its respective pins are working as input and output. For all the entry like selection of the items and for the entry of the prices the keypad is used which connected on PORT 3 of the 8051 and keypad is working as input for this microcontroller. The keypad is the 4x3 keypad which means that it has 4 rows and 3 columns. Another thing which is responsible for showing all type of messages is LCD. It is working as output for the microcontroller. The LCD is 16x2 which means that the LCD has 2 rows and in each row 16 columns are present. The final thing is the motor which runs when the coin entered has value equal to the actual value of the item. And it is also working as the output for the microcontroller. Also the pins of the controller cannot provide enough current to drive the DC motor so we use the H Bridge based motor drive like L298 or L293D.We can also use crystal oscillator of 11.0592MHz and two capacitors of 22pf with this controller. Also a rest button can also be attached with the hardware to reset the controller.

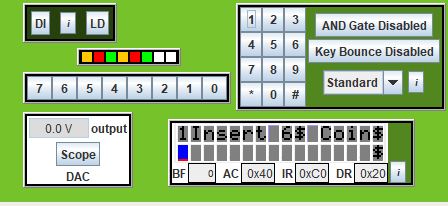
**Simulation of the project:**

**Step1:**

****

The LCD shows the total items present in the machine with their serial numbers and there price in front of them. For example for our project we have taken 4 objects like cake, CAN, Milk and tea. These items will be keep showing on the LCD until any item is selected using keypad.

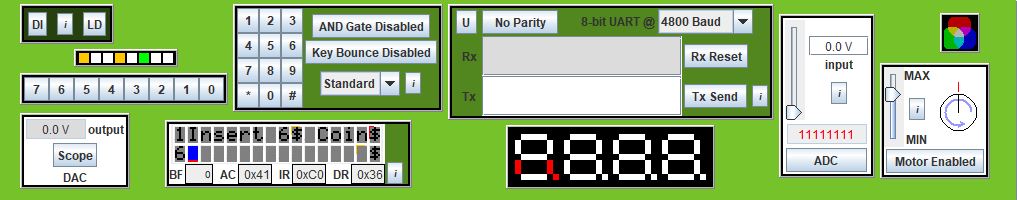
**Step2:**



For example if we select the 1st item then we press the 1 button on the keypad and then this message appear on the LCD which shows the serial number of the item and its price which we will enter using coin.

**Step3:**

The price of the item is entered using the keypad and after that the value is shown at the second line of the LCD. And then this value is compared with the actual value of the items. If both are equal then motor runs for specific time and return to the home items page otherwise it moves to home page without changing running the motor.



If coin value equal to item value then motor runs and then moves to the main page.

Else moves to the items page without running the motor.

**Code:**

; put data in RAM

0000| MOV 30H, #'1'

0003| MOV 31H, #'C'

0006| MOV 32H, #'A'

0009| MOV 33H, #'K'

000C| MOV 34H, #'E'

000F| MOV 35H, #' '

0012| MOV 36H, #'6'

0015| MOV 37H, #'$'

0018| MOV 38H, #' '

001B| MOV 39H, #'2'

001E| MOV 3AH, #'C'

0021| MOV 3BH, #'A'

0024| MOV 3CH, #'N'

0027| MOV 3DH, #' '

002A| MOV 3EH, #'5'

002D| MOV 3FH, #'$'

0030| MOV 40H, #'t'

0033| MOV 41H, #0 ; end of data marker

0036| MOV 42H, #'3'

0039| MOV 43H, #'M'

003C| MOV 44H, #'i'

003F| MOV 45H, #'l'

0042| MOV 46H, #'k'

0045| MOV 47H, #' '

0048| MOV 48H, #'4'

004B| MOV 49H, #'$'

004E| MOV 4AH, #' '

0051| MOV 4BH, #'4'

0054| MOV 4CH, #'T'

0057| MOV 4DH, #'e'

005A| MOV 4EH, #'a'

005D| MOV 4FH, #' '

0060| MOV 50H, #'3'

0063| MOV 51H, #'$'

0066| MOV 52H, #0 ; end of data marker

0069| MOV 53H, #'I'

006C| MOV 54H, #'n'

006F| MOV 55H, #'s'

0072| MOV 56H, #'e'

0075| MOV 57H, #'r'

0078| MOV 58H, #'t'

007B| MOV 59H, #0

007E| MOV 5AH, #' '

0081| MOV 5BH, #0

0084| MOV 5AH, #' '

0087| MOV 5BH, #' '

008A| MOV 5CH, #' '

008D| MOV 5DH, #' '

0090| MOV 5EH, #' '

0093| MOV 5FH, #' '

0096| MOV 60H, #' '

0099| MOV 61H, #' '

009C| MOV 62H, #' '

009F| MOV 63H, #' '

00A2| MOV 64H, #' '

00A5| MOV 65H, #' '

00A8| MOV 66H, #' '

00AB| MOV 67H, #' '

00AE| MOV 68H, #' '

00B1| MOV 69H, #' '

00B4| MOV 6AH, #0

Again\_do:

00B7| CLR P1.3 ; clear RS - indicates that instructions are being sent to the module

00B9| Acall FuncSet

00BB| Acall DispCon

00BD| Acall EntryMode

; send data

00BF| SETB P1.3 ; clear RS - indicates that data is being sent to module

00C1| MOV R1, #30H ; data to be sent to LCD is stored in 8051 RAM, starting at location 30H

loop:

00C3| MOV A, @R1 ; move data pointed to by R1 to A

00C4| JZ next ; if A is 0, then end of data has been reached - jump out of loop

00C6| CALL sendCharacter ; send data in A to LCD module

00C8| INC R1 ; point to next piece of data

00C9| JMP loop ; repeat

next:

00CB| Acall CursorPos

00CD| SETB P1.3

00CF| MOV R1, #42H ; data to be sent to LCD is stored in 8051 RAM, starting at location 30H

loop1:

00D1| MOV A, @R1 ; move data pointed to by R1 to A

00D2| JZ Next1 ; if A is 0, then end of data has been reached - jump out of loop

00D4| CALL sendCharacter ; send data in A to LCD module

00D6| INC R1 ; point to next piece of data

00D7| JMP loop1 ; repeat

Next1:

00D9| Call ScanKeyPad

00DB| Acall CursorPos1

00DD| SetB P1.3 ; RS=1 - Data register is selected.

00DF| Clr A

00E0| Mov A,R7

00E1| Call sendCharacter ;Display the key that is pressed.

00E3| SETB P1.3

00E5| MOV R1, #53H ; data to be sent to LCD is stored in 8051 RAM, starting at location 30H

loop2:

00E7| MOV A, @R1 ; move data pointed to by R1 to A

00E8| JZ go ; if A is 0, then end of data has been reached - jump out of loop

00EA| CALL sendCharacter ; send data in A to LCD module

00EC| INC R1 ; point to next piece of data

00ED| JMP loop2 ; repeat

go:

00EF| Mov A,R7

00F0| CJNE A,#'1',OV1 ;jump if A is not 1

00F3| Mov A,#'6'

00F5| SJMP OV4

OV1:

00F7| CJNE A,#'2',OV2 ;jump if A is not 2

00FA| Mov A,#'5'

00FC| SJMP OV4

OV2:

00FE| CJNE A,#'3',OV3 ;jump if A is not 3

0101| Mov A,#'4'

0103| SJMP OV4

OV3:

0105| CJNE A,#'4',OV4 ;jump if A is not 3

0108| Mov A,#'3'

OV4:

010A| Mov R6,A

;;;;;;;;;;;;;;;;;;;;;;;;;;;add space;;;;;;;;;;;

010B| Mov A, #' '

010D| Call sendCharacter

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

010F| Mov A, R6

0110| Call sendCharacter

0112| Mov A, #'$'

0114| Call sendCharacter

0116| Mov A, #' '

0118| Call sendCharacter

011A| Mov A,# 'C'

011C| Call sendCharacter

011E| Mov A, # 'o'

0120| Call sendCharacter

0122| Mov A, #'i'

0124| Call sendCharacter

0126| Mov A, #'n'

0128| Call sendCharacter

012A| Acall CursorPos

012C| SETB P1.3

012E| MOV R1, #5BH ; data to be sent to LCD is stored in 8051 RAM, starting at location 30H

loop3:

0130| MOV A, @R1 ; move data pointed to by R1 to A

0131| JZ h1 ; if A is 0, then end of data has been reached - jump out of loop

0133| CALL sendCharacter ; send data in A to LCD module

0135| INC R1 ; point to next piece of data

0136| JMP loop3 ; repeat

h1:

0138| Acall CursorPos

013A| Call ScanKeyPad

013C| SetB P1.3 ; RS=1 - Data register is selected.

013E| Clr A

013F| Mov A,R7

0140| Call sendCharacter ;Display the key that is pressed.

0142| Mov A,R6

0143| SUBB A,R7

0144| JNZ ED

0146| Setb p3.1

0148| clr p3.0

014A| Acall DELAY\_1s

014C| clr p3.1

014E| clr p3.0

ED:

0150| jmp Again\_do

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

DELAY\_1s:

0152| mov R1,#4

0154| RE2:MOV R2,#200

0156| RE1:MOV R3,#250

0158| RE: NOP

0159| NOP

015A| DJNZ R3,RE

015C| DJNZ R2,RE1

015E| DJNZ R1,RE2

0160| RET

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

CursorPos:

0161| Clr p1.3 ;RS

0163| setb P1.7

0165| setb P1.6

0167| Clr p1.5

0169| Clr p1.4

016B| call Pulse

016D| Clr P1.7

016F| Clr P1.6

0171| Clr p1.5

0173| Clr p1.4

0175| call Pulse

0177| CALL delay

0179| Ret

Clear:

017A| Clr p1.3 ;RS

017C| Clr P1.7

017E| Clr P1.6

0180| Clr p1.5

0182| Clr p1.4

0184| call Pulse

0186| Clr P1.7

0188| Clr P1.6

018A| Clr p1.5

018C| Clr p1.4

018E| call Pulse

0190| CALL delay

0192| Ret

CursorPos1:

0193| Clr p1.3 ;RS

0195| setb P1.7

0197| CLR P1.6

0199| Clr p1.5

019B| Clr p1.4

019D| call Pulse

019F| Clr P1.7

01A1| Clr P1.6

01A3| Clr p1.5

01A5| Clr p1.4

01A7| call Pulse

01A9| CALL delay

01AB| Ret

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

FuncSet:

; function set

01AC| CLR P1.7 ; |

01AE| CLR P1.6 ; |

01B0| SETB P1.5 ; |

01B2| CLR P1.4 ; | high nibble set ;;;;;

01B4| SETB P1.2 ; |

01B6| CLR P1.2 ; | negative edge on E

01B8| CALL delay ; wait for BF to clear

; function set sent for first time - tells module to go into 4-bit mode

; Why is function set high nibble sent twice? See 4-bit operation on pages 39 and 42 of HD44780.pdf.

01BA| SETB P1.2 ; |

01BC| CLR P1.2 ; | negative edge on E

; same function set high nibble sent a second time

01BE| SETB P1.7 ; low nibble set (only P1.7 needed to be changed)

01C0| SETB P1.2 ; |

01C2| CLR P1.2 ; | negative edge on E

; function set low nibble sent

01C4| CALL delay ; wait for BF to clear

01C6| Ret

;;;;;;;;;;;;;;;;;;;;;;; set to increment with no shift

EntryMode:

01C7| CLR P1.7 ; |

01C9| CLR P1.6 ; |

01CB| CLR P1.5 ; |

01CD| CLR P1.4 ; | high nibble set

01CF| Acall Pulse

01D1| SETB P1.6 ; |

01D3| SETB P1.5 ; |low nibble set

01D5| Acall Pulse

01D7| CALL delay ; wait for BF to clear

01D9| Ret

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

DispCon:

01DA| CLR P1.7 ; |

01DC| CLR P1.6 ; |

01DE| CLR P1.5 ; |

01E0| CLR P1.4 ; | high nibble set

01E2| Acall Pulse

01E4| SETB P1.7 ; |

01E6| SETB P1.6 ; |

01E8| SETB P1.5 ; |

01EA| SETB P1.4 ; | low nibble set

01EC| Acall Pulse

01EE| CALL delay ; wait for BF to clear

01F0| Ret

;;;;;;;;;;;;;;;;;;;;

Pulse:

01F1| SETB P1.2 ; |

01F3| CLR P1.2 ; | negative edge on E

01F5| Ret

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

sendCharacter:

01F6| MOV C, ACC.7 ; |

01F8| MOV P1.7, C ; |

01FA| MOV C, ACC.6 ; |

01FC| MOV P1.6, C ; |

01FE| MOV C, ACC.5 ; |

0200| MOV P1.5, C ; |

0202| MOV C, ACC.4 ; |

0204| MOV P1.4, C ; | high nibble set

0206| SETB P1.2 ; |

0208| CLR P1.2 ; | negative edge on E

020A| MOV C, ACC.3 ; |

020C| MOV P1.7, C ; |

020E| MOV C, ACC.2 ; |

0210| MOV P1.6, C ; |

0212| MOV C, ACC.1 ; |

0214| MOV P1.5, C ; |

0216| MOV C, ACC.0 ; |

0218| MOV P1.4, C ; | low nibble set

021A| SETB P1.2 ; |

021C| CLR P1.2 ; | negative edge on E

021E| CALL delay ; wait for BF to clear

delay:

0220| MOV R0, #50

0222| DJNZ R0, $

0224| RET

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

0225| ScanKeyPad: CLR P0.3 ;Clear Row3

0227| CALL IDCode0 ;Call scan column subroutine

0229| SetB P0.3 ;Set Row 3

022B| JB F0,Done ;If F0 is set, end scan

;Scan Row2

022E| CLR P0.2 ;Clear Row2

0230| CALL IDCode1 ;Call scan column subroutine

0232| SetB P0.2 ;Set Row 2

0234| JB F0,Done ;If F0 is set, end scan

;Scan Row1

0237| CLR P0.1 ;Clear Row1

0239| CALL IDCode2 ;Call scan column subroutine

023B| SetB P0.1 ;Set Row 1

023D| JB F0,Done ;If F0 is set, end scan

;Scan Row0

0240| CLR P0.0 ;Clear Row0

0242| CALL IDCode3 ;Call scan column subroutine

0244| SetB P0.0 ;Set Row 0

0246| JB F0,Done ;If F0 is set, end scan

0249| JMP ScanKeyPad ;Go back to scan Row3

024B| Done: Clr F0 ;Clear F0 flag before exit

024D| Ret

;--------------------------------------------------------------------------------

;---------------------------- Scan column subroutine ----------------------------

024E| IDCode0: JNB P0.4, KeyCode03 ;If Col0 Row3 is cleared - key found

0251| JNB P0.5, KeyCode13 ;If Col1 Row3 is cleared - key found

0254| JNB P0.6, KeyCode23 ;If Col2 Row3 is cleared - key found

0257| RET

0258| KeyCode03: SETB F0 ;Key found - set F0

025A| Mov R7,#'3' ;Code for '3'

025C| RET

025D| KeyCode13: SETB F0 ;Key found - set F0

025F| Mov R7,#'2' ;Code for '2'

0261| RET

0262| KeyCode23: SETB F0 ;Key found - set F0

0264| Mov R7,#'1' ;Code for '1'

0266| RET

0267| IDCode1: JNB P0.4, KeyCode02 ;If Col0 Row2 is cleared - key found

026A| JNB P0.5, KeyCode12 ;If Col1 Row2 is cleared - key found

026D| JNB P0.6, KeyCode22 ;If Col2 Row2 is cleared - key found

0270| RET

0271| KeyCode02: SETB F0 ;Key found - set F0

0273| Mov R7,#'6' ;Code for '6'

0275| RET

0276| KeyCode12: SETB F0 ;Key found - set F0

0278| Mov R7,#'5' ;Code for '5'

027A| RET

027B| KeyCode22: SETB F0 ;Key found - set F0

027D| Mov R7,#'4' ;Code for '4'

027F| RET

0280| IDCode2: JNB P0.4, KeyCode01 ;If Col0 Row1 is cleared - key found

0283| JNB P0.5, KeyCode11 ;If Col1 Row1 is cleared - key found

0286| JNB P0.6, KeyCode21 ;If Col2 Row1 is cleared - key found

0289| RET

028A| KeyCode01: SETB F0 ;Key found - set F0

028C| Mov R7,#'9' ;Code for '9'

028E| RET

028F| KeyCode11: SETB F0 ;Key found - set F0

0291| Mov R7,#'8' ;Code for '8'

0293| RET

0294| KeyCode21: SETB F0 ;Key found - set F0

0296| Mov R7,#'7' ;Code for '7'

0298| RET

0299| IDCode3: JNB P0.4, KeyCode00 ;If Col0 Row0 is cleared - key found

029C| JNB P0.5, KeyCode10 ;If Col1 Row0 is cleared - key found

029F| JNB P0.6, KeyCode20 ;If Col2 Row0 is cleared - key found

02A2| RET

02A3| KeyCode00: SETB F0 ;Key found - set F0

02A5| Mov R7,#'#' ;Code for '#'

02A7| RET

02A8| KeyCode10: SETB F0 ;Key found - set F0

02AA| Mov R7,#'0' ;Code for '0'

02AC| RET

02AD| KeyCode20: SETB F0 ;Key found - set F0

02AF| Mov R7,#'\*' ;Code for '\*'

02B1| RET

;--------------------------------- End of subroutines ---------------------------

02B2| Stop: Jmp $

End

**Flow Chart:**

Show 4 Items on LCD

With Price and Serial No

Start

Check for Keypad input

No Yes

Check either number 1,2,3,4

is pressed

Print item with price and wait for price entered

Yes yesZZ No

Yes

If entered amount is sufficient

Motor runs for some time

Yes

No

The starting of the code is from clearing the pin p1.3 which means that we are clearing RS pin of the LCD and we are going to pass the instructions to the LCD. After that we have called the function whose name is **FuncSet**. In this function the basic functionalities of the LCD are set like it is 16x2 or any other else. After that another function is called which is **DispCon** which is responsible for setting the position of cursor, start of line etc. After that the third function is called whose purpose is to set basic register which are necessary to print on the LCD. All these function have there are declared in code and are called here. After setting all these things now data is sent on the LCD is sent one by one character and finally displayed on LCD. The data is stored at prefixed location. In our case we have stored starting from 30H. So by this loop data starts sending character by character until 0 comes.

After printing data on first line we have to move the cursor to the second line and then we should start printing. So the address of the second **CursorPos** function. After execution of the function the cursor comes on the second line. Then we will print remaining items like item no 3 and 4 on the second line as we have printed on 1st line of LCD. The same method of the loop is used for this which is printing character by character on LCD. So at till at this stage our all items numbers with serial numbers and with price are printed on the LCD. On function which is very useful for printing the characters on LCD is **sendcharacter.** After printing all the items on LCD we again call a function nameas **ScanKeypad** which waits for the button pressed from the user **if user** presses any button for its choice it reads and stores that key in R7 register. And then we again call **sendcharacter** function to show that pressed key.

After the we print message on the screen like “1 Insert 5$ coin” but for this we have to know that for which serial number whose much dollars are assigned so for that we have to compare the pressed key with the dollar price and then we print that dollar amount with some text onto the LCD. For example if 1 is pressed then 5$ price indication should be printed on screen and for other keys other amount. If pressed key is not from 1 to 4 then do nothing. And for this comparison the CJNE is used.

Then the necessary messages are printed onto the screen with dollar amount. And all these characters are printed using **Sendcharacter** function. After printing respective messages on the LCD line 1 we have to move to the second line. So this is done using the **CursorPos** and again cursor of LCD moves to the first line.

After that then a function name as **CursorPos** is called which moves cursor from 1st line to 2nd line and then then the ScanKeypad is called which will take number from user as a entry of the coin. And then entered coin value is stored in the register R7 and by using **sendcharacter** function this key is printed on the LCD in the starting of second line. In last block the main things is that the above entered value is compared with the actual value of the and if entered value in R7 is equal to the actual price which is in R6 the motor runs for 1 second. The motor is connected on the P3.1. And if the entered price is not equal to the actual price then the motor will not run means that P3.1 will be cleared and Again whole process starts from starting from line one and LCD again starts printing item numbers.

**Project daetails:**

The whole project contain two big parts. The one big part is hardware part and the second big part is the software part. We make the software according to the hardware design and then we integrate both of these. Both parts have their own items to discuss which are these:

**Software design:**

There are many types of software available in market for the writing code for 8051. The most popular ones are KEIL, Edsim and Proteus. In this project we have used Edsim for software design. Edsim software has its own text editor where we can design our own software and we can debug, and can run it easily. The pic of Edsim text editor is added below.

The software is written in assembly language. There are many processes for writing code. First of all we have made code for LCD and then made code for the Keypad and in final step we have integrated all these code with integration of motor code in it.

**Hardware design:**

The hardware work for this project is also done in Edsim. The Edsim contains all the hardware components like LEDS, LCD, motor, seven segment, Keypad and apart from these all the registers including R1, R2, R3 and Accumulator Register. Also all the memory is also shown. And all these peripherals are connected with the 8051 microcontroller. The description hoe the hardware items are attached with the 8051 controller is also shown on the right side of the software the Picture of Edsim environment is also attached.