****

**JOMO KENYATTA UNIVERSITY**

**OF AGRICULTURE AND TECHNOLOGY**

**UNIT: MICROPROCESSORS I**

**UNIT CODE: EEE2406**

**TITLE: ATM REPORT**

**LAB III**

**MEMBERS**

**Davies Momanyi ENE211-0233/2016**

**Allan Mwirigi ENE211-0232/2016**

**Vitalis Ayacko ENE211-0025/2016**

**Thoenes Mbui ENE211-0016/2016**

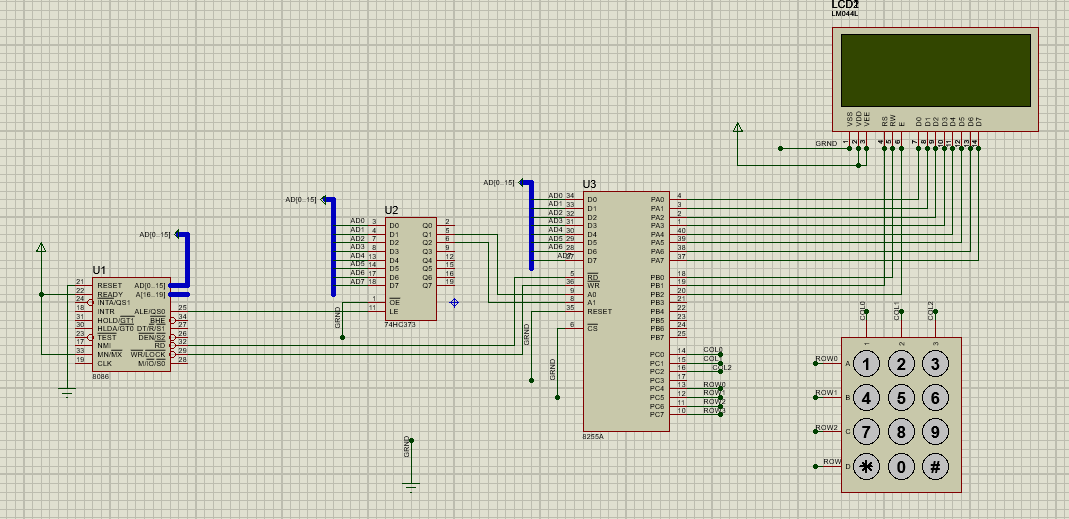
**Allan Kipkosgei ENE211-0249/2016**

**INSTRUCTIONS**

Design an assembly language program of a ten member Automatic Teller Machine with following capabilities.

1. Client identification password (four characters)
2. Request of withdrawal/deposit
3. Balance / denial of cash
4. Simulated using Proteus environment with LCD display.

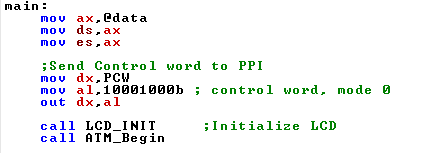
**CIRCUIT DIAGRAM**



In the above circuit, the 8086 microprocessor is connected to the 8255 PPI through a latch. The latch is required in order to reconcile the difference in operation speed between the microprocessor and the PPI. The 8086 is much faster than the 8255 therefore when it writes to the data bus, the data is only retained for very short period of time before it continues with other processes. The data needs to be held by the latch which contains flip flops. The data can then be transferred to/from the LCD and keypad through the PPI at their normal operation speed. The LCD is connected to PortA while the keypad is connected to PortB.

**IMPLEMENTATION**

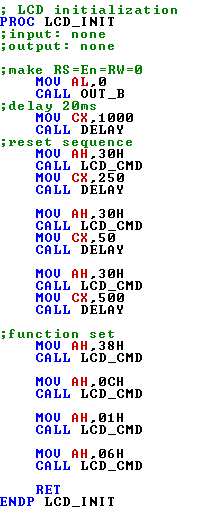
The data segment is first initialized to load all the defined variables. The control word **10001000B** is sent to the ControlPort (PCW) of the 8255 PPI. This control word sets PortC upper for input and the other ports for output.

****

The **LCD\_INIT** procedure is then called, which initializes the LCD before it is used. This is a necessary step specified by the device manufacturer whereby a series of commands are sent to the LCD. This procedure definition is shown on the next page.

The initialization consists of the following steps:

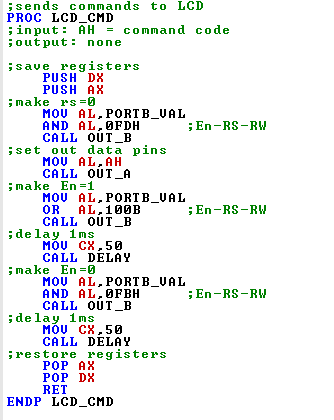
* Send command 0x30
* Delay 20ms
* Send command 0x30
* Delay 20ms
* Send command 0x30
* Delay 20ms
* Send Function set
* Display Clear command
* Set entry mode command



Each LCD command has a corresponding hex code that can be sent to the LCD. For instance:

|  |  |
| --- | --- |
| **Hex Code** | **Command to LCD instruction Register** |
| 01 | Clear display screen |
| 02 | Return home |
| 04 | Decrement cursor (shift cursor to left) |
| 06 | Increment cursor (shift cursor to right) |
| 05 | Shift display right |
| 07 | Shift display left |

These commands are used to control the operation of the LCD. To send these commands to the LCD, the **LCD\_CMD** procedure is called. Its implementation is as follows.

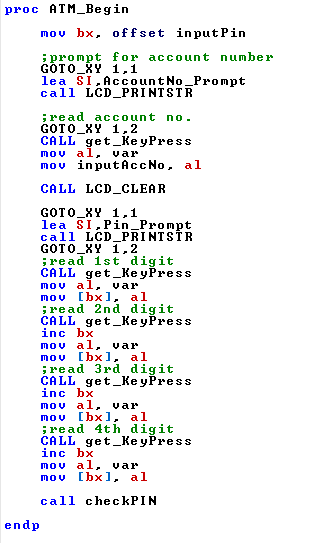


The command to be output is placed in the AL register and a series of required steps carried out for successful sending of the command. These steps are based on manufacturer specifications. The **OUT\_A** and **OUT\_B** procedures are called in order to send the command using the **OUT** instruction on the appropriate port.

Once the LCD has been initialized, the **ATM\_Begin** procedure is then called. This is the entry point of the ATM application. This application works in a number of steps:

1. Prompt for Account Number
2. Prompt for PIN
3. If the entered PIN matches the predefined PIN of the Account Number specified, display a menu of options for the user to transact (Deposit, Withdraw, Check Balance and Quit)

The ATM\_Begin procedure is defined as follows



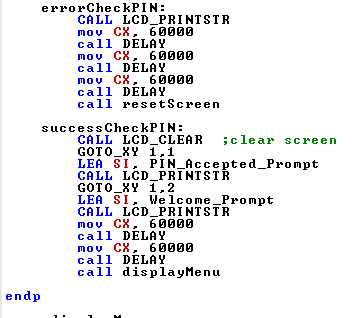
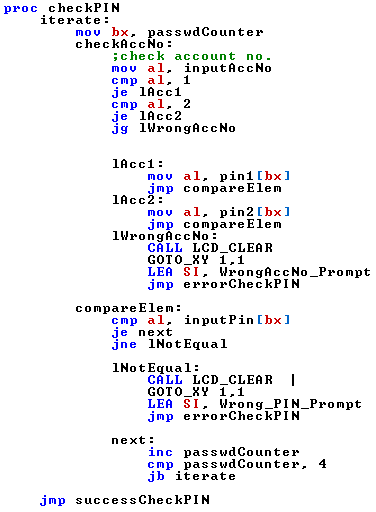
**inputPin** is an array meant to store the entered pin. Its offset address is placed in BX for later use. The user is prompted to enter their account number. The effective address of the message is loaded into SI and **LCD\_PRINTSTR** procedure called to display the message on the LCD. To get user input, the **get\_KeyPress** procedure is called which is used to read the input from the keypad and stores the value in a variable called **var.** Since **var** will be used severally in the program, this value is copied to another variable called **inputAccNo** to be used later.

Once the user has entered the account number and it has been stored, the screen is cleared and the user prompted to enter a PIN. In this case, **getKeyPress** is called four times. As the user enters the digits on the keypad, they are entered into register BX through register indirect addressing mode. BX holds the offset address of the **inputPin** array. BX is incremented with each entry to move to the next location where the digit read is to be stored. This results in an array of four digits which is to be evaluated.

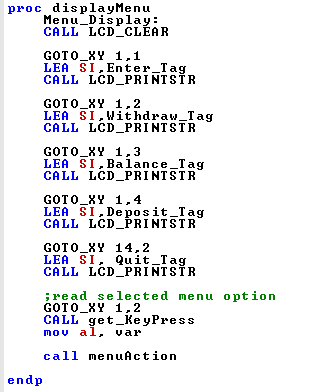
Once the PIN is read, **checkPin** procedure is called which evaluates whether the PIN matches the predefined PIN for the Account Number specified. This procedure is defined on the next page. It loops through all the elements of the array comparing each with the elements of the correct PIN array i.e. pin1 and pin2 for the first two accounts. **passwordCounter** is a variable with a value of 4. This value is assigned to register BX which will be used to index elements of the array as the loop runs. On each iteration of the loop, the account number is first checked then a conditional jump is used to move to a label based on the result of the check. For instance, if the user had entered account number as 1 (i.e. variable **inputAccNo** has value 1) then there is a jump to label **lAcc1** which sets a digit (from the correct PIN array using indexed addressing mode) in register AL, then a jump is made to do the comparison. Comparison is done in the **compareElem** label such that if the value in AL matches the value of the same index of the PIN that was entered by the user, then the next iteration proceeds. This loop runs four times based on the value of the **passwdCounter** variable. If not, then the loop is terminated and an error message is displayed. There is a delay to allow for the message to be read then the **resetScreen** procedure is called to reset the variables and return to the starting point of the application.

If an invalid account number was entered, the screen is cleared and an error message is displayed. There is a delay to allow for the message to be read then the **resetScreen** procedure is called to reset the variables and return to the starting point of the application.

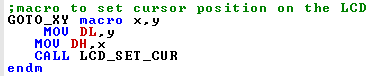
If all the digits of the PINs being compared match, then a success and welcome message is displayed. There is a delay to allow for the message to be read then the **displayMenu** procedure is called to show a list of options with which the user can transact.



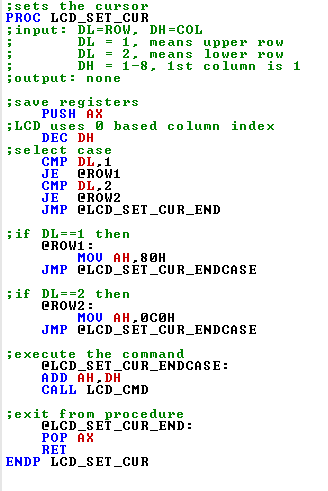
The **displayMenu** procedure is defined as follows



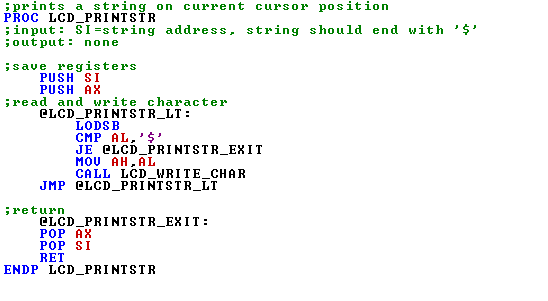
The screen is cleared and the **GOTO\_XY** macro is invoked. This macro is supplied with the x-position and y-position where the cursor is to be set on the LCD. These are placed in the DL and DH registers for use in the **LCD\_SET\_CUR** procedure which when called, sends the commands to LCD.



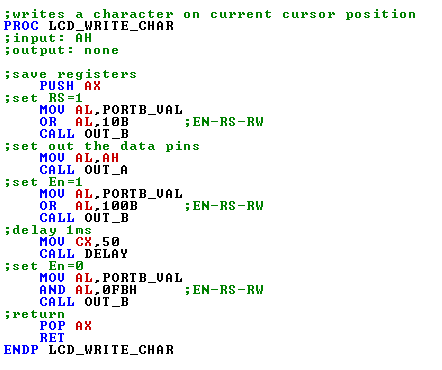
**LCD\_SET\_CUR** procedure uses values of register DL for the row and DH for the column. A value of 1 for DL corresponds to the first row while 2 corresponds to the second row. The procedure makes use of two commands to set the cursor position. **080H** moves the cursor to the beginning of the first line while **C0H** moves the cursor to the start of the second line.



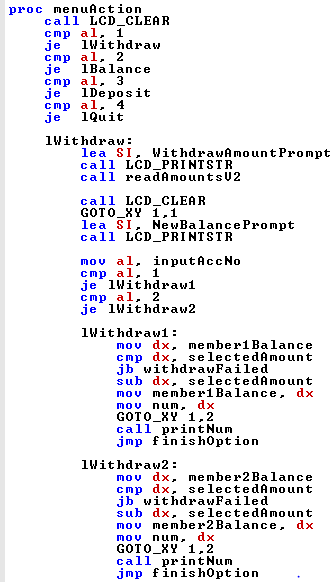
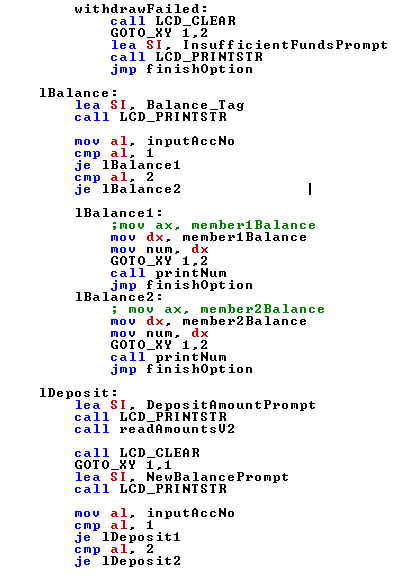
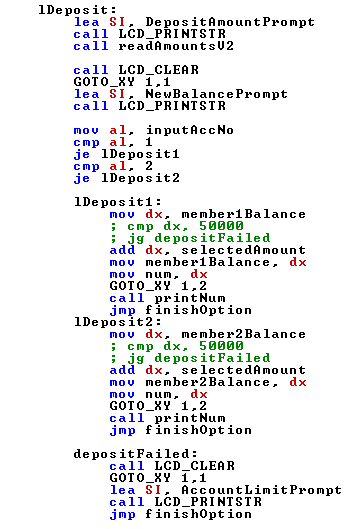
To display the string once the cursor has been set, the effective address of the string is first loaded into register SI and then the **LCD\_PRINTSTR** procedure is called.

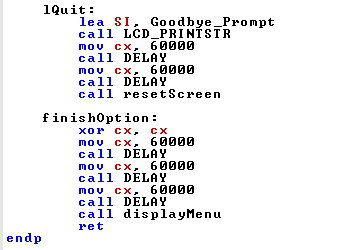


This procedure uses the LODSB instruction to load a byte from SI to the register AL. SI is the source operand and points to a sequence of bytes corresponding to the string. The byte position to be loaded is automatically incremented on each iteration of the LODSB instruction such that each character is placed into AL and displayed. AL is compared to ‘$’ on each iteration to check if the string has been fully covered then the loop ends. To display each character the **LCD\_WRITE\_CHAR** procedure is called with the character placed in AH register.



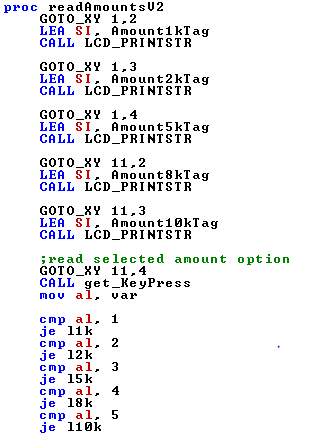
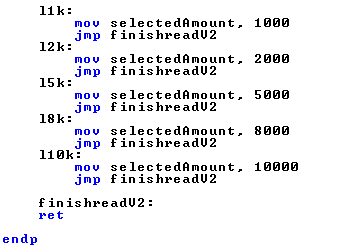
Once all the menu strings are displayed, the **getKeyPress** procedure is called which allows for user input. When the user selects an option on the menu by pressing the corresponding key on the keypad, the value of the key is stored in variable **var.** This value is then copied into the AL register for later use. The **menuAction** procedure is called to perform some action based on the option selected.



The above code details what is to be done when any of the options is selected. The option entered is stored in AL and is compared to preset values to determine which label to jump to i.e. lWithdraw, lDeposit, lBalance and lQuit.

In the case of lDeposit and lWithdraw, the user is first prompted to select the amount that they wish to transact. The **readAmountsV2** procedure is called.

This procedure displays a list of cash options that the user can select. The **get\_KeyPress** procedure is then called in order to read the user input. Depending on the key pressed, a certain value is copied to the **selectedAmount** variable. This is the value that the user intends to transact with.

The account number is then checked in order to proceed the correct balance for that account. For withdrawal, **selectedAmount** is subtracted from the balance and the new balance displayed. For deposit, **selectedAmount** is added to the balance and the new balance displayed. If **selectedAmount** exceeds the balance, then withdrawal is not possible and the user is prompted that there are insufficient funds for the transaction.

The Balance option displays the user’s current balance while the Quit option returns to the starting point of the application by calling the **resetScreen** procedure.

Display of the new balance is done by placing the value in the **num** variable and calling the **printNum** procedure. This procedure divides the number continuously to get each digit and displays it. Each digit is displayed on the screen hence the balance can be observed.

**PROGRAM OUTPUT**

The following images show the ATM application in various stages of operation.

