LCD and Hex keypad interfaced 8051 based 2-digit calculator using Assembly Language Program

Submitted by

1. SHOUVIK GHOSH #34900318031

2. SHIRSHENDU DAS #34900318033

3. SAYANDEEP MAHATA #34900318038

4. DIBENDU MONDAL #34900318063

5. AYUSH PODDAR #34900318068

Under the supervision of

**DR. PALASH DAS**

Assistant Professor

Department of Electronics and Communication Engineering,

Cooch Behar Government Engineering College, Cooch Behar, West Bengal





**ACKNOWLEDGEMENT**

It is our pleasure to acknowledge our supervisor Dr. Palash Das, Assistant Professor, Department of Electronics and Communication Engineering, Cooch Behar Government Engineering College, Cooch Behar, West Bengal, for being not only the source of encouragement, but also great resource of knowledge and information.

We shall remain indebted to him for the immense help we have received from him.

We would also like to thank all the faculty members, technical assistants and staff of the Department of Electronics and Communication Engineering, Cooch Behar Government Engineering College, for their kind and friendly cooperation extended to us.

We have no appropriate words to express sincere thanks to our family members for their continuous support and encouragement.

**CONTENTS**

|  |  |
| --- | --- |
| Topics | Page No. |
| Introduction | **4** |
| Components Used | **5** |
| Description of the components used  1.8051 MCU – AT89C52  2.Edsim  3.Matrix Keypad (4x3)  4.Hitachi hd44780u LCD  Module  5.Push Back Switch  6.LM7805 | **6-22**  **6-9**  **10-13**  **14-15**  **16-19**  **21**  **22** |
| Interfacing and program  1.Main program  2.Interfacing the keypad with 8051  microcontrollers  3.Interfacing the LCD module with 8051 microcontrollers | **23-24**  **24-32**  **33-36** |
| Code for mathematical operation | **35-37** |
| Limitations of our project | **38** |
| References and bibliography | **38** |

**INTRODUCTION**

Our Project of this course in this semester is LCD and Hex-Keypad Interfaced 8051 microcontroller based 2-digit calculator using Assembly language program. In this project we get to simulate a 2-digit calculator which perform basic calculations like addition, subtraction, division and multiplication of 2 digits. Here we have use Edsim51 simulator to create our assembly code and run it successfully.

This simulator is equipped with a Hitachi HD44780U LCD Module. There is also 3x4 keypad present in this simulator. At first, we have interfaced the LCD Module present with the keypad with our keypad interfacing code. Then we have written the code for the basic of the calculator. Accordingly, this project aims to develop the source code based on assembly language program. This Edsim51 simulator is a perfect go-through example to understand the working module 8051

microcontroller and hence has been quite helpful for this project.

**COMPONENTS USED**

The various components used in this project are as follows:

1. Edsim51
2. 8051 MCU – AT89C52
3. HITACHI HD44780U LCD module
4. Matrix Keypad (4x3)
5. Push Back Switch
6. LM7805
7. Crystal
8. 2 Capacitors
9. 9V Battery

* The code correspondingly used to carry out this project were run on edsim51 simulator.
* The logical diagrams corresponding to the project are based on the edsim51 simulator.

**CHAPTER – DESCRIPTION OF THE COMPONENTS USED**

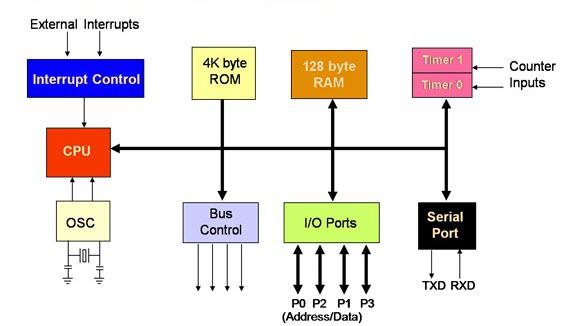
**8051 MCU – AT89C52**

* 8051 microcontroller is a 8-bit microcontroller.
* The features of 8051 microcontroller are as follows:

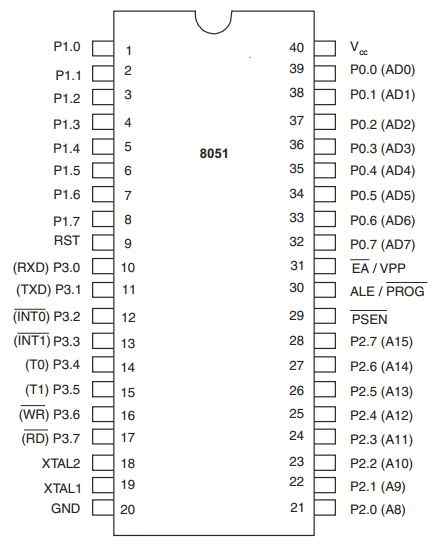
1. 4kB on chip program memory
2. 128 bytes on-chip data memory (RAM)
3. Four register banks
4. 128 user defined software flags
5. 8-bit bidirectional data bus
6. 16-bit unidirectional address bus
7. 32 general purpose registers each of 8-bit
8. 16-bit Timers (usually 2, but may have more or less)
9. Three internal and two external Interrupts
10. Four 8-bit ports, (short model have two 8-bit ports)
11. 16-bit program counter and data pointer
12. 8051 may also have a number of special features such as UARTs, ADC, Op-amp, etc.

* 8051 microcontroller family members has 3 more types of microcontrollers – 8051, 8052 and 8031

**The following illustration shows the block diagram of an 8051 microcontroller −**



**The pin diagram of 8051 microcontroller is shown is the figure below –**



**The pin description along with the pin numbering is being discussed in the following table:**

|  |  |  |
| --- | --- | --- |
| SL.NO. | PIN NUMBERS | DESCRIPTION |
| 1. | Pin 1 - 8 | These pins are known as Port 1. This port doesn’t serve any other functions. It is internally pulled up, bi-directional I/O port. |
| 2. | Pin 9 | It is a RESET PIN, used to reset the microcontroller to its initial values. |
| 3. | Pin 10 - 17 | Known as Port 3, these ports serves some functions like interrupts, timer input, control signals, serial communication signals RxD and TxD, etc |
| 4. | Pin 18 & 19 | These pins are used for interfacing an external crystal to get the system clock. |
| 5. | Pin 20 | Provides the power supply to the circuit. |
| 6. | Pin 21 - 28 | Known as Port 2 , they served as I/O port. |
| 7. | Pin 29 | PSEN pin which is used to read a signal from external program memory. |
| 8. | Pin 30 | EA pin which is used to enable/disable the external memory interfacing |
| 9. | Pin 31 | ALE pin which is used to demultiplex the address-data signal of port |
| 10. | Pin 32 - 39 | Known as Port 0 |
| 11. | Pin 40 | Used to provide power supply to the circuit |

* An addressing made is a way to locate a target data, which is also called as operand. The 8051 family of microcontrollers allows five types of addressing modes for addressing operands. They are as follows:

    i) Immediate Addressing

   ii) Register Addressing

  iii) Direct Addressing

  iv) Register-Indirect Addressing

 v) Indexed Addressing

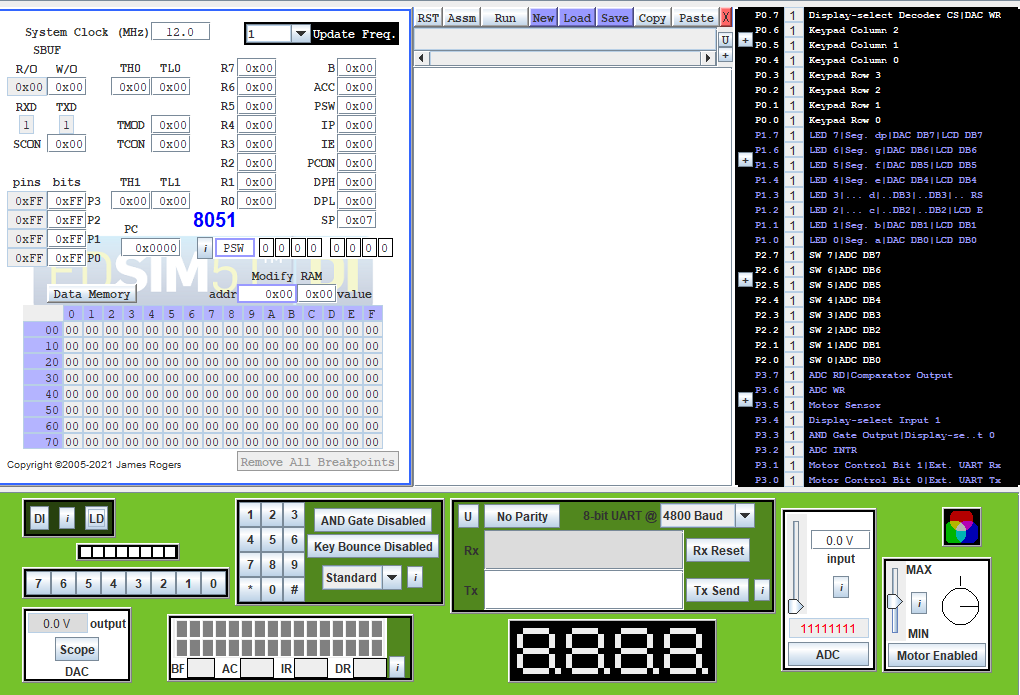
* Coming to the operand part of the instruction, it defines the data being processed by the instructions. The operand can be any of the following:

1. No Operand
2. Data Value
3. I/O Port
4. Memory Location
5. CPU Register

* Based on the operations they perform, all the instructions in 8051 microcontroller instruction set are divided into five groups:

1. Data Transfer Instruction
2. Arithmetic Instruction
3. Logical Instructions
4. Boolean or bit Manipulation Instructions
5. Program Branching Instructions.

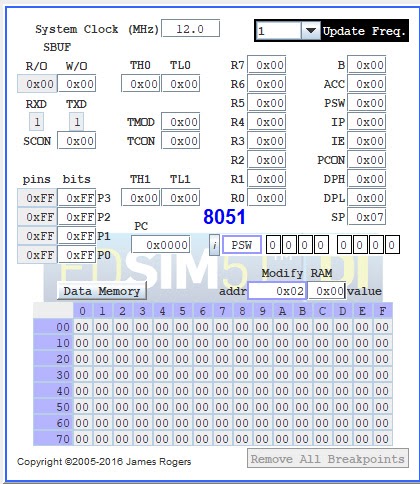
**EDSIM51**



* Edsim51 is a widely used simulation software for programming of 8051.
* The various parts of edsim51 are as follows:

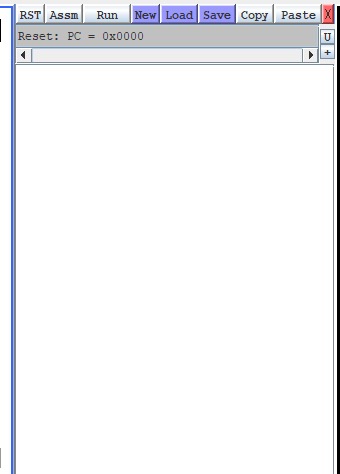
1. Registers and Internal RAM
2. Programming Section
3. Port Connections
4. Sensors and Actuators

* REGISTERS AND INTERNAL RAM-

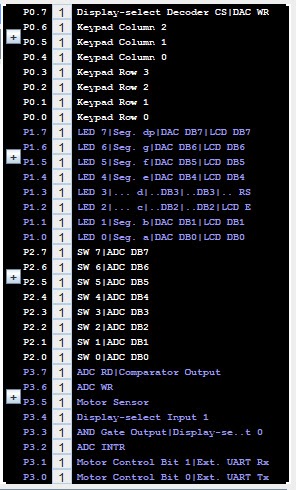
1. The upper parts depict different registers such as general-purpose registers (R0-R7), SFRs.
2. A memory map of internal RAM which is 128 bytes meaning 128 locations, starting from 0x00, 0x01,….,0x0F (ROW-1) . Similarly, there are 8 rows.

* PROGRAMMING SECTIONS-

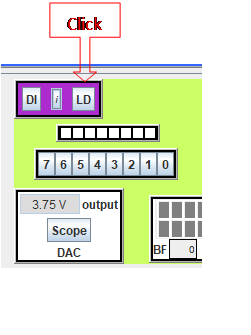
1. Once the program is written, save the program into the PC’s local storage.
2. Program also can be loaded from the memory.
3. After typing the program, run it.
4. Check the register values and memory locations.
5. Click RST to come back to the edit part.

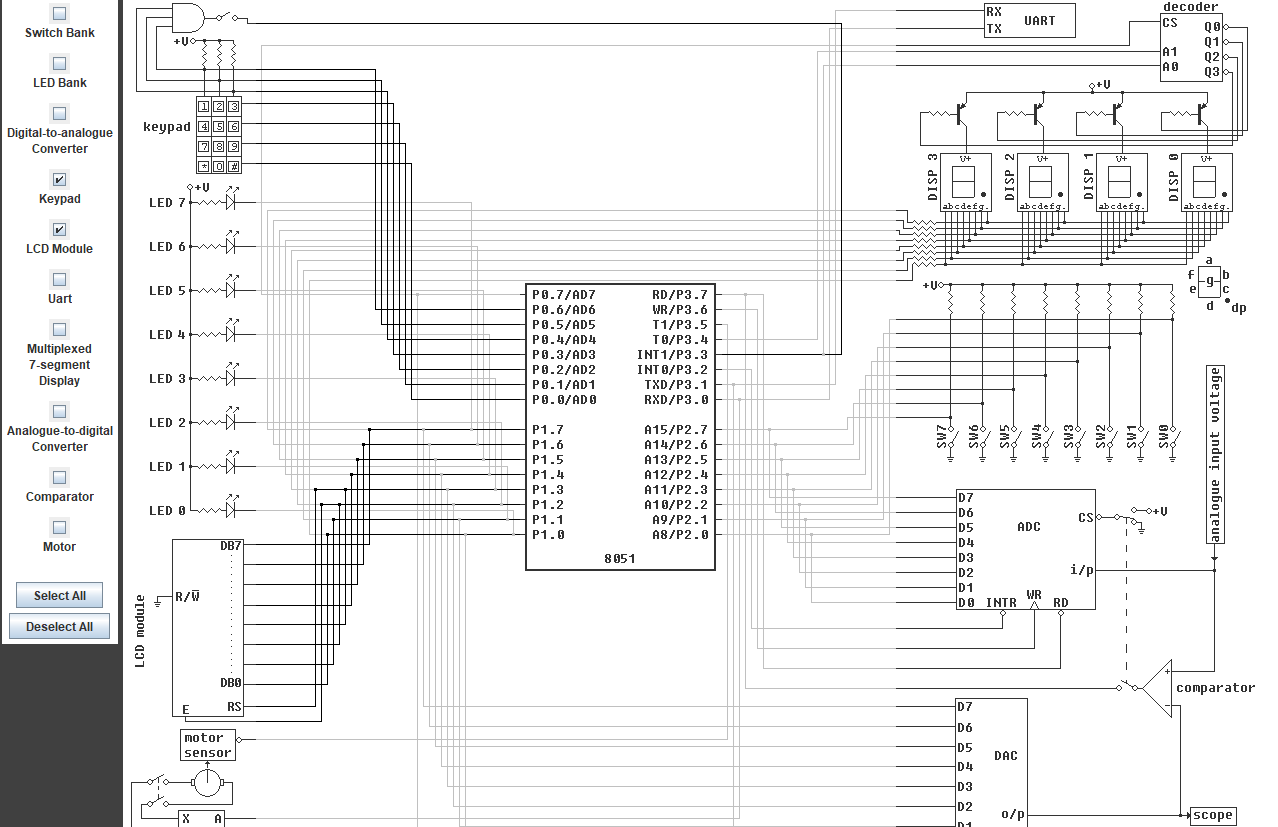


**PORT INTERFACE-**



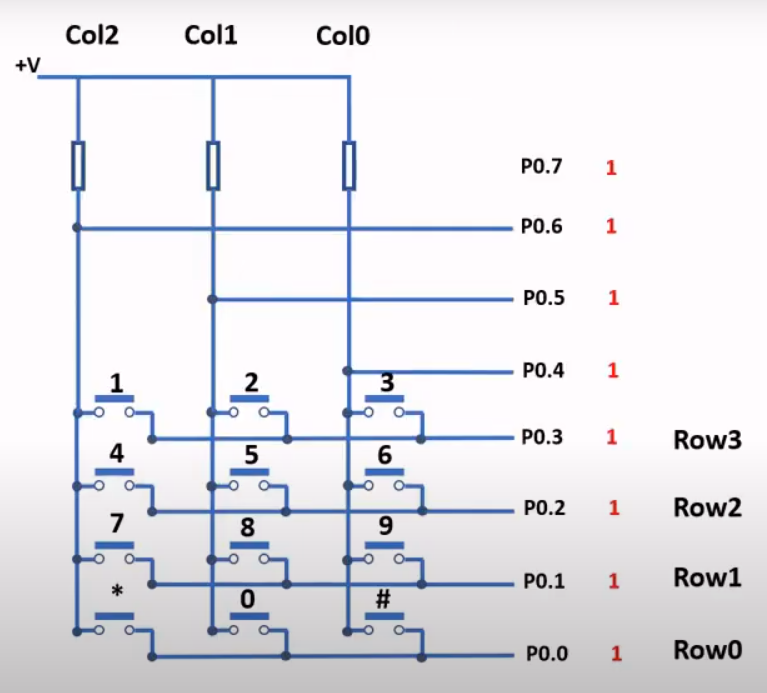
Logic Diagram





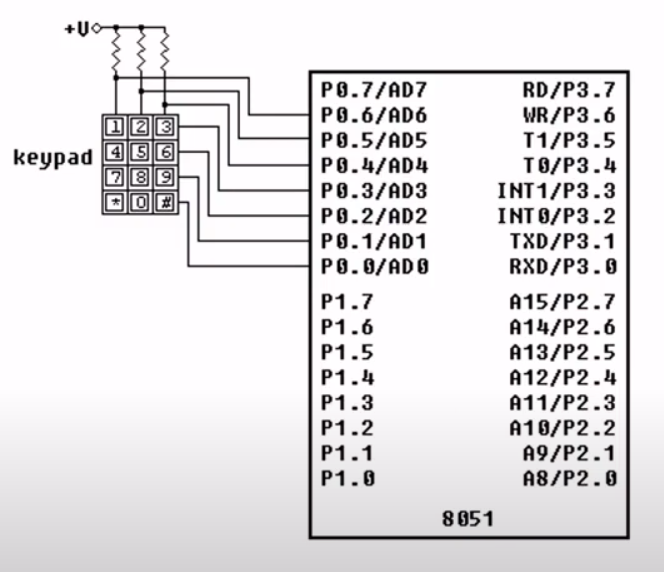
**MATRIX KEYPAD(4X3)**

* Most of the applications of [embedded systems](http://www.edgefx.in/importance-of-embedded-systems-in-automobiles-with-applications/) require keypads to take the user inputs, especially in case where an application requires a greater number of keys.
* With simple architecture and easy interfacing procedure, matrix keypads are replacing normal push-buttons by offering more inputs to the user with the lesser I/O pins.
* A matrix keypad consists of a set of push button or switches which are arranged in a matrix format of rows and columns.
* These keypads are available in configurations like 3×4 and 4×4 based on the application it is implemented for. Internal diagram of this matrix keypad is shown in the below figure-



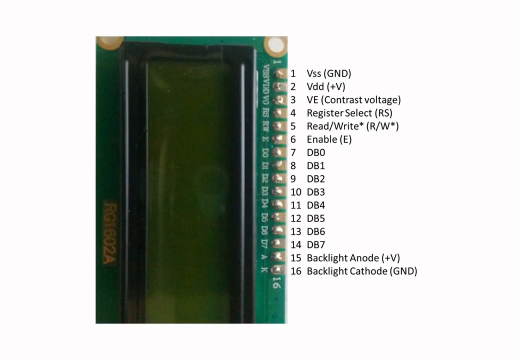
* Matrix keypad can be connected to the microcontroller in numerous ways or techniques, but the fundamental logic is same as making the columns as input and the rows as output.
* So, in order to detect the key pressed from the keypad, the row lines have to be made low one by one and to read the columns.

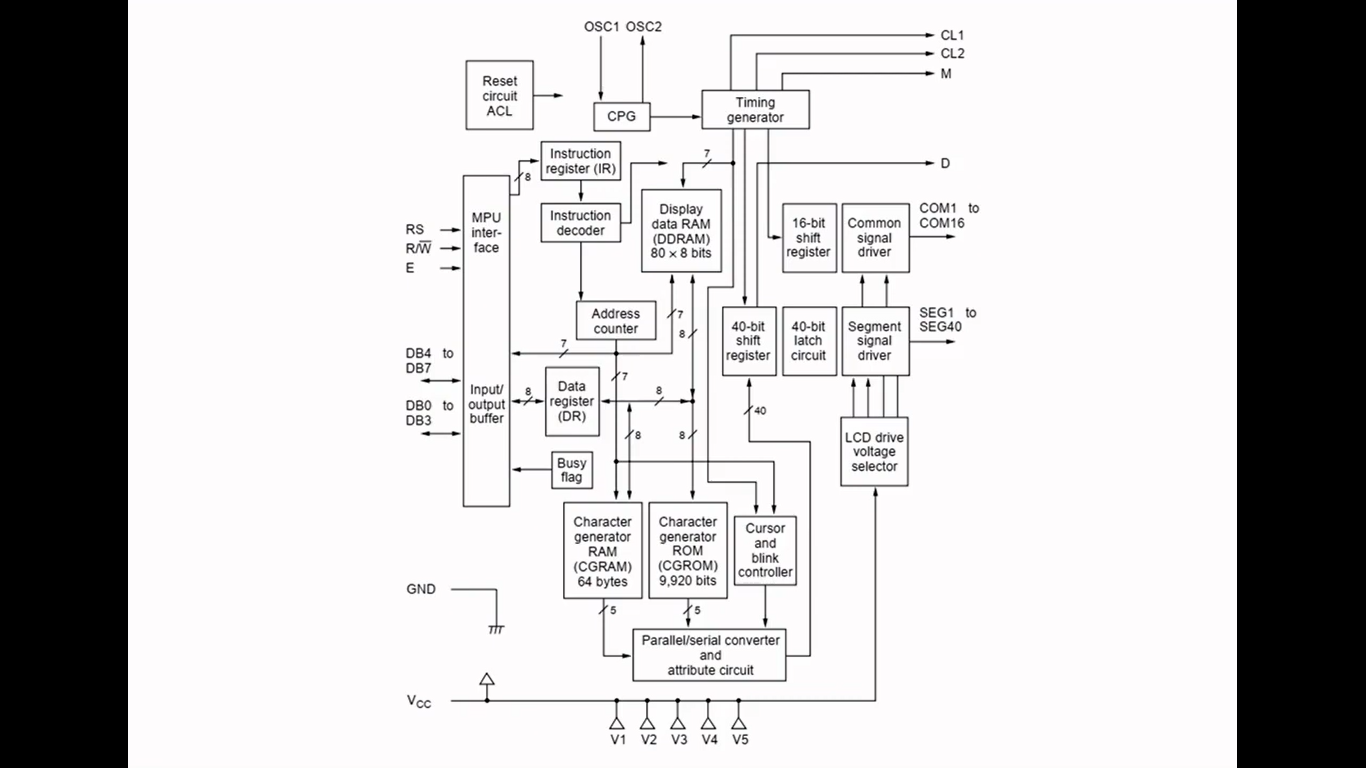
MATRIX KEYPAD INTERFACE WITH MICROCONTROLLER-



1. Make sure that the all the rows of port 1 high in order to give the signal to microcontroller when any key is pressed.
2. The working of the keypad goes like this: If any of the keys in row1 of the matrix keypad is pressed, the corresponding column line will give low and similarly if the second key is pressed in row1, then the column line 2 will give low. This process is repeated for all the rows.
3. When the circuit is powered and any key is pressed, then corresponding pins of the port 1 get enabled. Depending on the program in the microcontroller, it displays the number.

**HITACHI HD44780U LCD MODULE:**

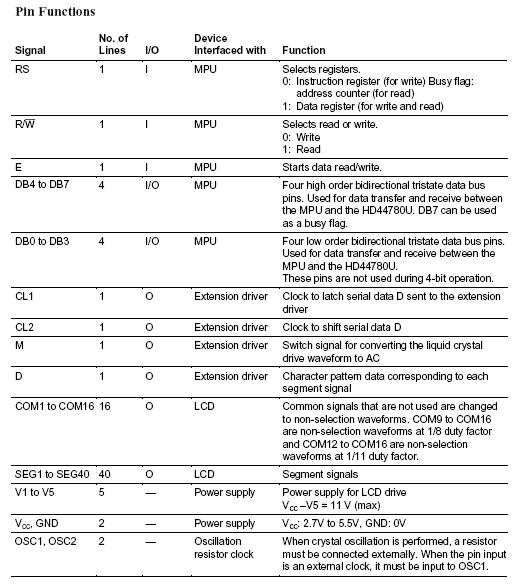
* The HD44780U dot-matrix LCD controller and driver LSI displays alphanumeric, kana characters and symbols.
* It can be configured to drive a dot-matrix LCD under the control of a 4 or 8 bit microprocessor / microcontroller.
* The low power supply (2.7V to 5.5V) of the HD44780U is suitable for any portable battery-driven product requiring low power dissipation.
* FEATURES –
* 5 × 8 and 5 × 10 dot matrix possible.
* Low power operation support:  2.7 to 5.5V.
* Corresponds to high speed MPU bus interface - 2MHz.
* Supports wide range of instruction functions.
* Automatic reset ckt that initialises the controller/driver after power on.

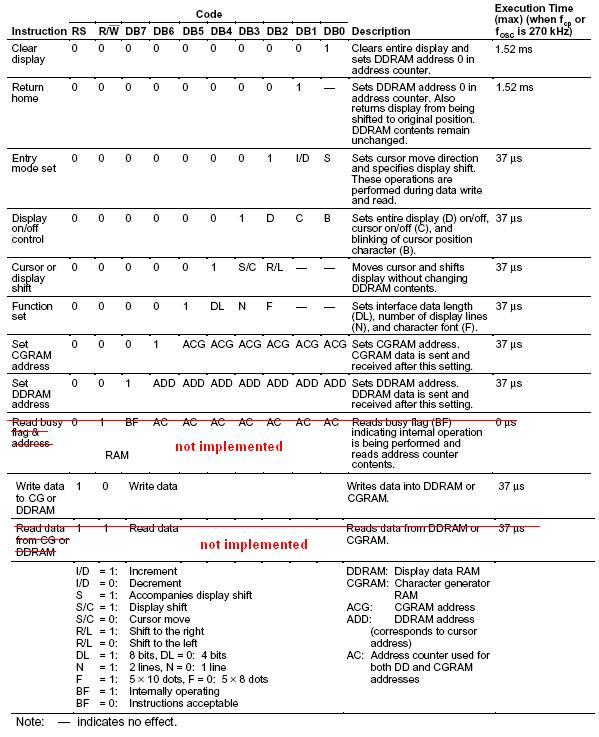
**The block diagram of HD44780U is shown in the given diagram:**

* **4-bit Mode:**The LCD module is a simulation of the Hitachi HD44780 and is interfaced to the 8051 in 4-bit mode. P1.7 through P1.4 are connected to DB7 through DB4, while P1.3 is connected to the register-select pin and P1.2 is connected to the enable pin. Notice the read-write pin is connected to ground - the module can only be written to.
* **8-bit Mode:**By default, as stated above, the module is interfaced in 4-bit mode. However, the lower four data bits (DB3 through DB0) are also available (on P1.3 through P1.0). If the user wishes to write to the module in 8-bit mode, RS and E should be remapped to other port pins.

**The pin description of the LCD provided in edsim51**

**is given below:**

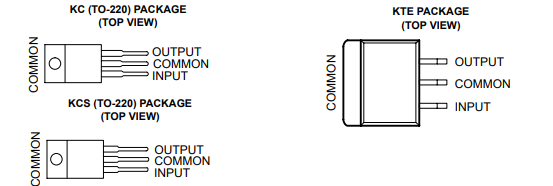


**The instruction set of LCD Module is discussed below-**

**PUSH BACK SWITCH**

* A simple Push Back Switch is a type of switch which consists of a simple electric mechanism or air switch mechanism to turn something on or off.
* Depending on model they could operate with momentary or latching action function.
* The button itself is usually constructed of a strong durable material such as metal or plastic.
* Push back switches come in a range of shapes and sizes.
* Push back switches are used throughout industrial and medical applications and are also recognisable in everyday life.
* For uses within the industrial sector, push back switches often part of a bigger system and are connected through mechanical linkage.
* This means that when a button is pressed it can cause another button to release.

**LM7805**

****

* This series of fixed-voltage integrated-circuit voltage regulators is designed for a wide range of applications.
* These applications include on-card regulation for elimination of noise and distribution problems associated with single-point regulations.
* Each of these regulators can deliver up to 1.5 A of output current.
* The internal current-limiting and thermal-shutdown features of these regulators essentially make them immune to overload.
* In addition to use as fixed-voltage regulators, these devices can be used with external components to obtain adjustable output voltages and currents, and also can be used as the power-pass element in precision regulators.
* Features of LM78705:

1. 3 terminal regulators.
2. Output Current up is 1.5A
3. Internal-thermal overload production
4. High power-dissipation capacity
5. Internal short-circuit current limiting.
6. Output Transistor SAFE-Area Compensation.

**CHAPTER – INTERFACING AND PROGRAM**

**MAIN PROGRAM**

Org 0000h

E Equ P1.2 ; pin P1.2 is assigned for Enable

RS Equ P1.3 ; pin P1.3 is assigned for Register select

N1 Equ 30H ;stores first digit of first operand

N2 Equ 32H ;stores second digit of first operand

N3 Equ 34H ;stores first digit of second operand

N4 Equ 36H ;stores second digit of second operand

N5 Equ 40H ;stores result after calculation

N6 Equ 42H ;stores quotient of result

N7 Equ 44H ;stores remainder of result

; -------------------------------------- Main -----------------------------------------;

**Main:**

**Clr** RS ; RS=0 - Instruction register (I/R) is selected.

Call MemInit ;Calls the memory initialisation subroutine and

clears all the garbage value.

;----------------------------- Instructions Code ------------------------------------;

**Call** FuncSet ; Function set (selecting the 4-bit mode)

**Call** DispCon ; Turn display and cusor on/off

**Call** EntryMode ; Entry mode set - shift cursor to the right

;---------------------------------- Scan for the keys --------------------------------;

**Next:**

**Call** ScanKeyPad ; calls the scan keypad subroutine

**SetB** RS ; RS=1 - Data register is selected.

**Clr** A

**Mov** A,R7 ; move the pressed key to accumulator

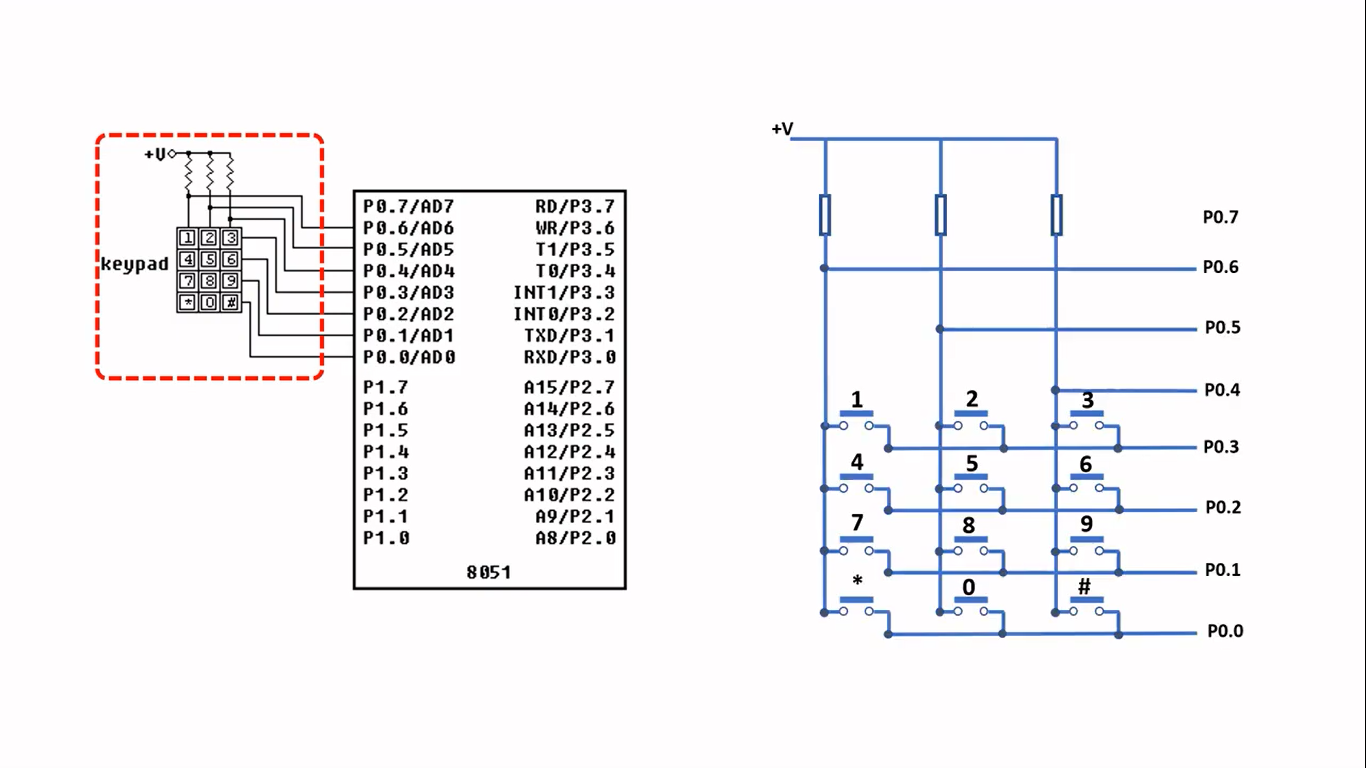
**Call** SendChar ; Display the key that is pressed.

**Call** operand ;stores the 1st and 2nd operand in R3 & R5

**EndHere:** **Jmp** Next

;------------------------------------ \*End Of Main\* --------------------------------;

**INTERFACING THE KEYPAD WITH 8051 MICROCONTROLLER**

**The interface between the keypad and the 8051 microcontroller is shown in the diagram below:**

In edsim, the keypad uses the port 0 of the 8051 microcontrollers out of which pin P0.0 to P0.6 is being used and P0.7 is left unused. Pin p0.0 - pin p0.3 are connected to row0, row1, row2, row3 respectively & P0.4 – P0.6 are connected to Col0, Col1, Col2 respectively.

Initially all pins from p0.0 to pin 0.6 are set high (logic 1). When scanning the Row0, pin P0.0 is made low (logic 0), and let’s assume that key # is pressed. So, on pressing #, it will short the Row0 and Col0 and simultaneously pin P0.4 will automatically go low. Hence the microcontroller will detect that p0.4 is low i.e. The key pressed is #.

**The assembly language code for the interfacing of the keypad is discussed below:**

; Col2 Col1 Col0

; +----+----+----+

; | 1 | 2 | 3 | Row3

; +----+----+----+

; | 4 | 5 | 6 | Row2

; +----+----+----+

; | 7 | 8 | 9 | Row1

; +----+----+----+

; | \* | 0 | # | Row0

; +----+----+----+

;-------------------------------------- Scan Row ----------------------------------------;

**ScanKeyPad:**

;Scan Row3

CLR P0.3 ;Clear Row3

CALL IDCode0 ;Call scan column subroutine

SetB P0.3 ;Set Row 3

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

;Scan Row2

CLR P0.2 ;Clear Row2

CALL IDCode1 ;Call scan column subroutine

SetB P0.2 ;Set Row 2

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

;Scan Row1

CLR P0.1 ;Clear Row1

CALL IDCode2 ;Call scan column subroutine

SetB P0.1 ;Set Row 1

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

;Scan Row0

CLR P0.0 ;Clear Row0

CALL IDCode3 ;Call scan column subroutine

SetB P0.0 ;Set Row 0

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

JMP ScanKeyPad ;Go back to scan Row3

**Done:** Clr F0 ;Clear F0 flag before exit

Ret

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

**IDCode0:**

JNB P0.4, KeyCode03 ;If Col0 Row3 is cleared - key found

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

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

RET

**KeyCode03:**

SETB F0 ;Key found - set F0

Mov R7,#'3' ;Code for '3'

;---------------- Checks the value of R6, and according to that it jumps to required subroutine--------------;

cjne R6,#00H, next1 ; If R6 is 0, execute the next command, otherwise jump to next1

Mov N1,#03H ; stores this digit as digit 1 of operand 1 in N1 memory location

inc R6

ret

**next1:**

cjne R6,#01H, next2 ; If R6 is 1, execute the next command, otherwise jump to next2

Mov N2,#03H ; stores this digit as digit 2 of operand 1 in N2 memory location

inc R6

ret

**next2:**

cjne R6,#02H, next3 ; If R6 is 2, execute the next command, otherwise jump to next3

Mov N3,#03H ; stores this digit as digit 1 of operand 2 in N3 memory location

inc R6

ret

**next3:**

cjne R6,#03H, $ ; If R6 is 3, execute the next command, otherwise do nothing

Mov N4,#03H ; stores this digit as digit 2 of operand 2 in N4 memory location

RET

**KeyCode13:** SETB F0 ; Key found - set F0 flag

Mov R7,#'2' ; Code for '2'

cjne R6,#00H, next4

Mov N1,#02H

inc R6

ret

**next4:** cjne R6,#01H, next5

Mov N2,#02H

inc R6

ret

**next5:** cjne R6,#02H, next6

Mov N3,#02H

inc R6

ret

**next6:** cjne R6,#03H, $

Mov N4,#02H

RET

**KeyCode23:** SETB F0 ;Key found - set F0

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

cjne R6,#00H, next7

Mov N1,#01H

inc R6

ret

**next7:** cjne R6,#01H, next8

Mov N2,#01H

inc R6

ret

**next8:** cjne R6,#02H, next9

Mov N3,#01H

inc R6

ret

**next9:** cjne R6,#03H, $

Mov N4,#01H

RET

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

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

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

RET

**KeyCode02:** SETB F0 ;Key found - set F0

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

cjne R6,#00H, next10

Mov N1,#06H

inc R6

ret

**next10:** cjne R6,#01H, next11

Mov N2,#06H

inc R6

ret

**next11:** cjne R6,#02H, next12

Mov N3,#06H

inc R6

ret

**next12:** cjne R6,#03H, $

Mov N4,#06H

Ret

**KeyCode12:** SETB F0 ;Key found - set F0

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

cjne R6,#00H, next13

Mov N1,#05H

inc R6

ret

**next13:** cjne R6,#01H, next14

Mov N2,#05H

inc R6

ret

**next14:** cjne R6,#02H, next15

Mov N3,#05H

inc R6

ret

**next15:** cjne R6,#03H, $

Mov N4,#05H

RET

**KeyCode22:** SETB F0 ;Key found - set F0

Mov R7,#'4' ;Code for '4'

cjne R6,#00H, next16

Mov N1,#04H

inc R6

ret

**next16:** cjne R6,#01H, next17

Mov N2,#04H

inc R6

ret

**next17:** cjne R6,#02H, next18

Mov N3,#04H

inc R6

ret

**next18:** cjne R6,#03H, $

Mov N4,#04H

RET

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

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

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

RET

**KeyCode01:** SETB F0 ;Key found - set F0

Mov R7,#'9' ;Code for '9'

cjne R6,#00H, next19

Mov N1,#09H

inc R6

ret

**next19:** cjne R6,#01H, next20

Mov N2,#09H

inc R6

ret

**next20:** cjne R6,#02H, next21

Mov N3,#09H

inc R6

ret

**next21:** cjne R6,#03H, $

Mov N4,#09H

RET

**KeyCode11:** SETB F0 ;Key found - set F0

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

cjne R6,#00H, next22

Mov N1,#08H

inc R6

ret

**next22:** cjne R6,#01H, next23

Mov N2,#08H

inc R6

ret

**next23:** cjne R6,#02H, next24

Mov N3,#08H

inc R6

ret

**next24:** cjne R6,#03H, $

Mov N4,#08H

RET

**KeyCode21:** SETB F0 ;Key found - set F0

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

cjne R6,#00H, next25

Mov N1,#07H

inc R6

ret

**next25:** cjne R6,#01H, next26

Mov N2,#07H

inc R6

ret

**next26:** cjne R6,#02H, next27

Mov N3,#07H

inc R6

ret

**next27:** cjne R6,#03H, $

Mov N4,#07H

RET

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

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

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

RET

**KeyCode00:** SETB F0 ; Key found - set F0

Mov R7,#'=' ; Code for '#'

Clr A

Mov A,R7

Call SendChar

;--------------------------------- R4 stores the number of times the key '\*' is pressed-------------------------------;

cjne R4,#01H, OP2 ; If R4=1, jump to ADDITION subroutine

Call ADDITION

**OP2:**

cjne R4,#02H, OP3 ; If R4=2, jump to SUBTRACTION subroutine

Call SUBTRACTION

**OP3:**

cjne R4,#03H, OP4 ; If R4=3, jump to MULTIPLICATION subroutine

Call MULTIPLICATION

**OP4:**

Call DIVISION ; Jump to DIVISION subroutine

RET

**KeyCode10:** SETB F0 ;Key found - set F0

Mov R7,#'0' ;Code for '0'

cjne R6,#00H, next28

Mov N1,#00H

inc R6

Ret

**next28:** cjne R6,#01H, next29

Mov N2,#00H

inc R6

Ret

**next29:** cjne R6,#02H, next30

Mov N3,#00H

inc R6

Ret

**next30:** cjne R6,#03H, $

Mov N4,#00H

RET

**KeyCode20:** SETB F0 ;Key found - set F0

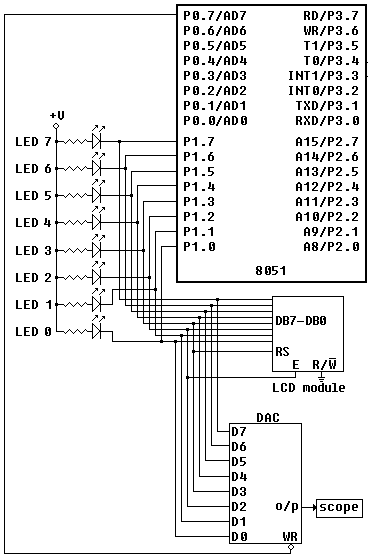
Mov R7,#'\*' ;Code for '\*'

inc R4

RET

**INTERFACING THE LCD MODULE WITH 8051 MICROCONTROLLER**

**The interface of the LCD module and 8051 microcontrollers**

**is shown in the diagram below:**

**The assembly language codes behind the working of the LCD is being discussed below:**

;------------------------------ \*Code for LCD\* -------------------------------;

**FuncSet:** ; Function set (for interfacing lcd in 4-bit mode)

; P1.4 - P 1.7 controls both DB0-DB3 and DB4-DB7

Clr P1.7

Clr P1.6

SetB P1.5

Clr P1.4 ; (DB4)DL = 0 -> puts LCD module into 4-bit mode.

Call Pulse ; negative edge on E, the module reads the data lines

DB7 - DB4

Call Delay ; wait for BF (busy flag) to clear

Call Pulse ; negative edge on E

SetB P1.7 ; P1.7=1 (N=1) -> turns on 2 lines of display

Clr P1.6 ; P1.6=0 (F=0) -> font set to 5 x 8 dots

Clr P1.5

Clr P1.4

Call Pulse

Call Delay

Ret

;----------------------------- Display on/off control --------------------------;

**DispCon:** ; The display and cursor are turned on

Clr P1.7 ; high nibble set

Clr P1.6 ;|

Clr P1.5 ;|

Clr P1.4 ;|

Call Pulse

; lower nibble set

SetB P1.7 ;

SetB P1.6 ; Sets entire display ON

SetB P1.5 ; Cursor ON

SetB P1.4 ; Cursor blinking ON

Call Pulse

Call Delay ; wait for BF to clear

Ret

;-------------------------- Entry mode set (4-bit mode) ----------------------;

;--------------Set to increment the DDRAM address by one and cursor shifted to the right------------;

**EntryMode:**

Clr P1.7 ;| high nibble set

Clr P1.6 ;|

Clr P1.5 ;|

Clr P1.4 ;|

Call Pulse

; lower nibble set

Clr P1.7 ; P1.7 = '0'

SetB P1.6 ; P1.6 = '1'

SetB P1.5 ; P1.5 = '1' ; I/D=1 -> Increment ; I/D=0 -> Decrement

Clr P1.4 ; P1.4 = '0'

Call Pulse

Call Delay ; wait for BF to clear

Ret

;----------------------------------- SendChar -----------------------------------;

;Display the key that is pressed------------;

**SendChar:** Mov C, ACC.7 ; | high nibble set

Mov P1.7, C ; |

Mov C, ACC.6 ; |

Mov P1.6, C ; |

Mov C, ACC.5 ; |

Mov P1.5, C ; |

Mov C, ACC.4 ; |

Mov P1.4, C ; |

Call Pulse

Mov C, ACC.3 ; | low nibble set

Mov P1.7, C ; |

Mov C, ACC.2 ; |

Mov P1.6, C ; |

Mov C, ACC.1 ; |

Mov P1.5, C ; |

Mov C, ACC.0 ; |

Mov P1.4, C ; |

Call Pulse

Call Delay ; wait for BF to clear

Ret

;-------------------------------Pulse subroutine --------------------------------;

**Pulse:**

SetB E ; P1.2 is connected to 'E' pin of LCD module

Clr E ; i.e. The LCD module will read the data lines (DB7-DB4)

Ret ; on the falling edge of the signal

;-------------------------------- Delay Subroutine -----------------------------;

**Delay:**  Mov R0, #50

Djnz R0, $

Ret

**CODE FOR MATHEMATICAL CALCULATION.**

* Finally we come to the main part of Designing where we intend to use the calculator.
* The assembly language code for the various operations is shown below-

ADDITION: **Mov** A,R3

**Add** A,R5

**Call** ResultBits ; Extracting two digits from the final result

**Call** Ascii1 ; Converts the 1st HEX digit into ASCII

**Call** Ascii2 ; Converts the 2nd HEX digit into ASCII

**jmp** $

SUBTRACTION: Mov A,R3 ;Code for subtraction

Subb A,R5

Jc Negative ;jump if result of calculation is negative

Call Positive

Positive: Call ResultBits ;printing for positive result

Call Ascii1

Call Ascii2

jmp $

Negative: Mov R7,#'-' ;for displaying -ve character

Clr A

Mov A,R7

Call SendChar

Clr A

Mov A,R5 ;calculating sub result in case of -ve

Subb A,R3

Inc A

Call ResultBits ;displaying -ve result

Call Ascii1

Call Ascii2

jmp $

MULTIPLICATION: Mov A,R3 ;code for multiplication

Mov B,R5

Mul AB

Call ResultBits ;displaying results

Call Ascii1

Call Ascii2

jmp $

DIVISION: Mov A,R3 ;code for divison

Mov B,R5

Div AB

Call ResultBits ;displaying results

Call Ascii1

Call Ascii2

jmp $

;--------------- Extracting two digits from the calculation result ---------------;

ResultBits: mov N5,A ;moves result to N5

mov B,#0AH

Div AB

Mov N6,A ;moves quotient of result to N6

Mov N7,B ;moves remainder of result to N7

ret

* Since the result obtained from our calculator is in Hexadecimal form so we need to convert it to ASCII Form for the LCD display.

Ascii1 : MOV A, N6 ; Store first digit into A

CLR C

SUBB A, #0AH ; Check if digit is greater than 10

MOV A, N6

JC SKIP1

ADD A, #07H ; Add 07 if >09

Call SendChar

ret

SKIP1: ADD A, #30H ; Else add only 30h for 0-9

Call SendChar

Ascii2 : MOV A, N7 ; Store second digit into A

CLR C

SUBB A, #0AH ; Check if digit is greater than 10

MOV A, N7

JC SKIP2

ADD A, #07H ; Add 07 if number is >09

Call SendChar

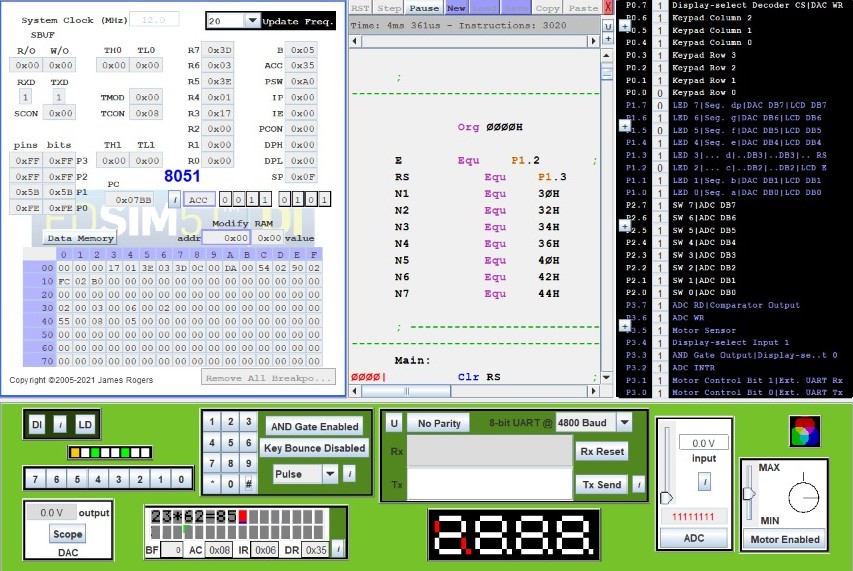
ret

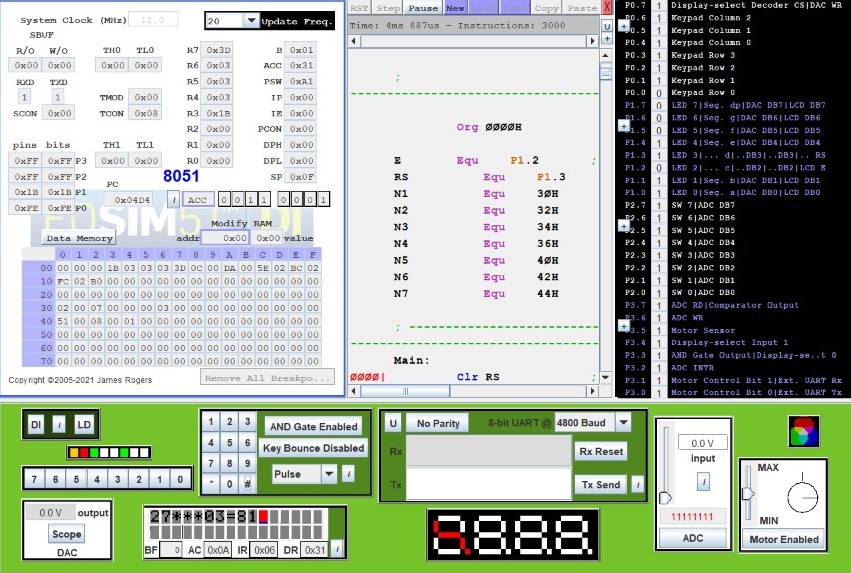
SKIP2: ADD A, #30H ; Else add only 30h for 0-9

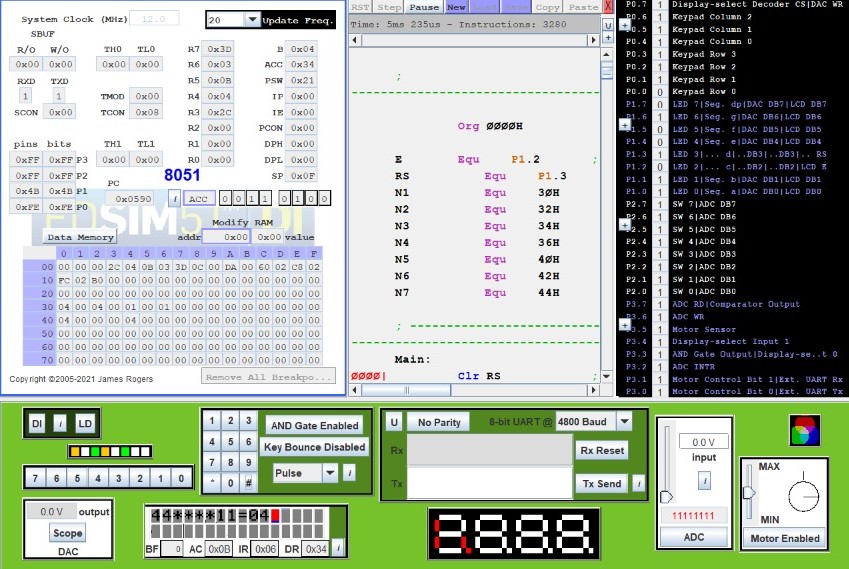
Call SendChar

End

**Simulation results**







**LIMITATIONS OF OUR PROJECT**

There are certain limitations in our project:

Firstly, we are restricted to only 8-bit operations.

Secondly, the results displayed are only integral values and no fractional result is displayed or any fractional data cannot be provided as input.

**REFERENCES AND BIBLIOGRAPHY**

* Microprocessor architecture, Programming, and Applications - Ramesh Gaonkar
* The 8051 Microcontroller - AYALA
* [https://www.edsim51.com/examples.html#prog4\_b](https://www.edsim51.com/examples.html)
* <https://www.edsim51.com/examples/lcd.asm>
* [https://www.edsim51.com/simInstructions.html#lcdModule](https://www.edsim51.com/simInstructions.html)