LAB FILE

of

EE1531:
MICROCONTROLLER & EMBEDDED SYSTEMS LAB

Name:
Registration Number:
Programme: B. Tech. (Electrical & Electronics Engineering)
Section:
Batch:
Session: July-Nov, 2016
General Instructions

❖ Mobile Phones and other Electronic Gadgets must be kept in the Designated Rack available at the Lab Entrance.

❖ Maintain Proper Discipline in the Laboratory. Avoid unnecessary Movement and Discussions.

❖ Always wear Shoes in the Laboratory.

❖ No Food Items are allowed inside the Laboratory. Student should keep the Work Area Clean and Well Organised.

❖ Do Not wear Metallic Objects like Watches, Bracelets etc. during Laboratory Sessions.

❖ Do Not plug in External Storage Devices to the Equipment and Workstations Without Permission of the Lab Technician. Do Not Delete Files / Software from the Workstations in the Lab.

Safety Precautions

❖ Keep yourself aware regarding Location and Use of Available Safety Equipment.

❖ Perform Laboratory work only with the Permission of the Lab In-charge and/or Supervisor.

❖ Do not use the Equipment that you are Unfamiliar with.

❖ Read the Lab manual carefully and connect the Circuit accordingly.

❖ Double Check the circuit for Proper Connection and Polarity. Get the circuit verified by the Faculty and Lab Technician before Switching on the power.

❖ Do not make Circuit Changes or perform any wiring when Power is On. First apply Low Voltages to check proper functionality of circuits and thereafter increase Voltages to the required level.

❖ If you smell anything burning, immediately disconnect the supply and call the Lab technician and Faculty to examine the circuit.

❖ Always remember to turn off the Equipment properly once your Work is done.
Laboratory Examination Evaluation Pattern

**Continuous Evaluation:** 60 %

- ✔ Documentation : 15
- ✔ Preparation and Observation : 10
- ✔ Regularity and Performance : 15
- ✔ Assignment Submission : 10
- ✔ Internal Viva : 10

**End Semester Evaluation:** 40 %

- ✔ Procedure with circuit, format : 15
- ✔ Experiment Conduction : 10
- ✔ Calculations : 05
- ✔ Viva Voce : 10

Department of Electrical Engineering
List of Experiments

Experiment 1: Study of 8051 based Microcontroller Kit (ESA MCB51 Kit) and Keil Software.
Experiment 2: Write assembly language programs to transfer the data from one memory location to another memory location using different addressing mode.
Experiment 3: Write assembly language programs to transfer a data block from one place to another place in the memory of Microcontroller 8051.
Experiment 4: Write assembly language programs to search a number in block available in memory of Microcontroller 8051.
Experiment 5: Write assembly language programs to sort a data block available in memory of Microcontroller 8051.
Experiment 6: Write assembly language programs for mathematical operation (addition and subtraction) using Microcontroller 8051.
Experiment 7: Write and perform programs for mathematical operation (multiplication and division) using Microcontroller 8051.
Experiment 8: Write and perform programs of mathematical operations on BCD Numbers.
Experiment 9: Write and perform programs of the various number system based code conversion operation.
Experiment 10: Write and simulate programs for Delay Generation, Timer & Counter operations.
Experiment 11: Write and simulate programs for interrupt operation on ESA51 Microcontroller Kit.
Experiment 12: Write and perform programs to display the data on LCD Screen interfaced with ESA51 Microcontroller Kit.
Experiment 13: Write and perform programs of various logic operations on ESA51 Microcontroller Kit.
Experiment 14: Write assembly language programs to interface & run a stepper motor at various speed using ESA51 Microcontroller Kit.

References:
- Vender Instructional Manual.
Experiment 1
Introduction to ESAMCB – 51 kit and Keil software

Aim: Study of 8051 based Microcontroller Kit (ESA MCB 51 Kit) and Keil Software.

Equipment used: ESA MCB 51 Kit, Computer System.

Software Used: Keil uVision Software.

Description:

ESA-MCB51 kit: ESA MCB 51 is a development board and supports a wide variety of 8051 (with on chip ROM) compatible 8-bit micro controllers. The 8051 family of micro controllers are extensively used for embedded and real-time applications. ESA MCB 51 is designed to be a general-purpose development board for Single Chip MCU applications that may be used as an instructional learning aid and also as a development tool in R&D labs in industries. ESA MCB 51 development board is built around Atmel AT89C51ED2 / RD2 microcontroller that has 64Kbytes of on-chip program memory.

ESA MCB 51 CAPABILITIES

- The power full on-chip flash monitor provides communication with KeilμVision Debugger.
- Executes the User Programs at full speed or Debug the Program using Single Step and Break Point facilities available in Keil μ Vision Debugger
- Examine/Modify the contents of CPU registers and memory contents.
- On-Board LCD (16x2), interfaced to port lines.
- All the 32 I/O (Port) lines are terminated on 4 different 10-pin connectors.
- 24 port lines (i.e. P0, P1 and P2) are terminated on a 26-pin connector.
- All the CPU Signals are terminated on a 44-pin J4 connector for system expansion.
- On-Board ISP support for On-Chip flash programming.
SPECIFICATIONS:

□ PROCESSOR
✓ AT 89C51ED2 / RD2 operating at 11.0592 MHz

□ PROCESSOR FEATURES
✓ ON-CHIP MEMORY
   ➢ CODE MEMORY: 64K Bytes of flash.
✓ DATA MEMORY:
   ➢ 256 Bytes of RAM.
   ➢ 1792 Bytes of XRAM.
   ➢ 2K Bytes of EEPROM.

□ ON-CHIP PERIPHERALS
✓ 3 nos of 16-bit Timers/Counters.
✓ Watch Dog Timer.
✓ Programmable Counter Array (PCA) on Port1 ie; PWM and Capture & Compare.
✓ SPI (Serial Peripheral Interface) on Port1.
✓ Full duplex enhanced UART.

□ INTERRUPTS
✓ Nine sources of interrupt (both external and internal).
✓ Two External interrupts INT0 and INT1 are provided with push button switches;
✓ These can also be used as general-purpose switches.

INTERFACE SIGNALS
✓ CPU:
   ➢ All the CPU signals available on a 40-pin connector site.
✓ I/O (Port) Lines:
   ➢ P0, P1 & P2 Ports are available on a 26-pin connector, compatible to ESA Interface modules.
✓ LCD:
   ➢ LCD compatible signals are available on a 16 Pin flow strip connector.
✓ SERIAL I/O:
   ➢ On-Chip UART signals are available through MAX 232 on a RJ-11 connector.
Keil μVision4 software

μVision4 is an IDE (Integrated Development Environment) that helps you write, compile, and debug embedded programs. It encapsulates the following components:

✓ A project manager.
✓ A make facility.
✓ Tool configuration.
✓ Editor.
✓ A powerful debugger.
✓ Supports about 400 Microcontrollers.

Configuring the μVision4

✓ Open Keil μvision 4 software.
✓ Open the Flash menu and select configure flash tools.
   ➢ Configure flash tools with the following:
   a) Output: Create Hex file.
   b) Debug: Use simulator; Limit Speed to Real Time
   c) Target: Select XTAL 11.0592 MHz.
✓ Open the Project menu and select new project. A Create New Project window will appear with a Save in the box where the project files are to be saved. Browse until the chosen disk and folder are found and type a project name into the File name box. A “project name.UV2” has now been created in the folder.
✓ The Files window will now show a “Target 1” folder, and a “Select Device for ‘Target 1’ ” window will appear. Browse the Atmel folder and select AT89C51ED2. Click OK and then NO to the “Copy Standard 8051 startup code…” question in the pop-up box.
✓ Expanding the “Target 1” folder in the Files window shows that it contains a subfolder named “Source Group 1.”
✓ Open the File menu and select New. Type the assembly program as a text file and save it as .asm file in the same folder where the project was saved.

Click on target and then double click on source group. A “Add files to Group Source group 1” window will appear. Add the saved/required .asm file.

Procedure to assemble, debug and run the program:

✓ Select the .asm file to be assembled.
Select the Build target from the Project menu.

The program will be assembled and warnings & errors if any will be indicated. A Hex file will also be created and saved in the same directory where the project was saved.

In case of any errors in the program, target will not be created. Rectify the errors, save and rebuild the target.

If the target is created, open the Debug menu, select start/stop debug session. Debugging session begins. Click off the pop-up window which tells that we are running a 2K evaluation version of μVision.

Change the memory contents if required.

Open the Debug menu and select Run.

Verify the register / memory / peripheral contents to test the program.

If the desired results are not obtained insert break points in the program or select start/stop debug session from debug menu and then Step option. Program will be executed step by step i.e. one instruction at a time. Go on pressing F11 and verify register/memory contents after every instruction to find the bugs in the program.

Viewing memory/changing memory contents
After the debug session begins open View menu and select memory window. Memory window with option for four memory windows Memory 1 to Memory 4 appears. Select any of the memory windows 2 to 4.

For viewing different memory contents, valid memory address types are:

- Internal data (RAM) : d: 00H to d:OFFH
- Code data beginning at C: 0000H
- External data (RAM) beginning at X: 0000H

To change the memory contents double click on the required memory cell and enter the required data byte.

Note
- Once in debug mode, μVision will not respond to most commands except debugger commands. Hence stop debugging before editing or reassembling the program under test. The debugger will not take any notice of changes made to the program while debugging.
- When more than one assembly program files are added to the same source group, while assembling and debugging, right click on the ASM files select options for “filename.asm”.

Department of Electrical Engineering
Uncheck the options “Include in target build”, “Always build” and “Link publics only” for all other files except the one which is to be assembled/debugged.

- Save your files in C:\MCLAB\Batch Name (A1, B2, C1 etc):Roll No.
- Add all the exercises (program files) of a particular week to the same project.

**Assembler directives**

Assembler directives are pseudo operations which inform the assembler what to do with the mnemonics that it is to assemble. Some of the commonly used assembly directives are:

- **ORG**: This directive can be used to put (starting address) the code and data anywhere in the memory. Normally ORG 0000H is used to start the program at 0000H.
- **EQU**: This directive is used to define a constant without occupying a memory location. It associates a constant value with a data label so that wherever the label appears in the program, its constant value is substituted for the label. EQU directive makes the program more readable.
  
  **Example**: Count EQU 25
  MOV R4, #Count

- **DB**: (Define byte) This directive is used to define the ‘8’ bit data. The numbers can be in binary, decimal, hex, decimal or ASCII formats. The assembler stores the bytes starting at the address specified in the code memory.
  
  **Example**: ORG 0300H
  TABLE: DB 30H, 31H, 32H, 33H, 34H
  Five data bytes defined are stored in code memory starting at 0300H.

  Similarly DW directive can be used to define a word (16 bit) data.

- **END**: This directive indicates the assembler that it is the end of the program. It is the last line of an 8051 program. Any source code after END directive is ignored by the assembler.

  Note: Anything that follows a semicolon (;) in a line of program is ignored by the assembler. Hence ‘ ;’ can be used to include comments in the program.

Exercise 1: Explain the procedure to write the assembly code in KEIL Software.

Assignment 1: Compare the features of Microcontroller 8051, 8052 & 8031.
Experiment 2
Programs of Data transfer & Addressing Modes

Aim: Write assembly language programs to transfer the data from one memory location to another memory location using different addressing mode.

Equipment used: Computer System.

Software Used: Keil uVision Software.

Description:

MOV instruction is the basic instruction for data transfer. The format of MOV instruction is as follows:

\[ \text{MOV Destination, Source} \]

The all possible way of use MOV instructions are as follow:

i. \( \text{MOV} \ @Rn, \ #\text{immediate} \)
ii. \( \text{MOV} \ @Ri, \ A \)
iii. \( \text{MOV} \ @Ri, \ direct \)
iv. \( \text{MOV} \ A, \ #\text{immediate} \)
v. \( \text{MOV} \ A, \ @Ri \)
vi. \( \text{MOV} \ A, \ direct \)
vii. \( \text{MOV} \ A, \ Rn \)
viii. \( \text{MOV} \ bit, \ C \)
ix. \( \text{MOV} \ C, \ bit \)
x. \( \text{MOV} \ dest\_direct, \ src\_direct \)
xi. \( \text{MOV} \ direct, \ #\text{immediate} \)

The MOVC instruction moves a byte from the code or program memory to the accumulator.

i. \( \text{MOVC} A, \ @A+DPTR \)
ii. \( \text{MOVC} A, \ @A+PC \)

The MOVX instruction transfers data between the accumulator and external data memory. External memory may be addressed via 16-bits in the DPTR register or via 8-bits in the R0 or R1 registers. When using 8-bit addressing, Port 2 must contain the high-order byte of the address.

i. \( \text{MOVX} @Ri, \ A \)
ii. \( \text{MOVX} A, \ @DPTR \)

Addressing Mode:
An "addressing mode" refers to how you are addressing a given memory location.

In summary, the addressing modes are as follows, with an example of each:

**Immediate Addressing** \( \text{MOV} A, \#20h \)
**Direct Addressing**  \[\text{MOV A,30h}\]

**Indirect Addressing**  \[\text{MOV A,@R0}\]

**External Direct**  \[\text{MOVX A,@DPTR}\]

**Code Indirect**  \[\text{MOVC A,@A+DPTR}\]

Each of these addressing modes provides important flexibility.

**Immediate Addressing**

Immediate addressing is so-named because the value to be stored in memory immediately follows the operation code in memory. That is to say, the instruction itself dictates what value will be stored in memory.

For example, the instruction:
\[\text{MOV A,#20h}\]

This instruction uses Immediate Addressing because the Accumulator will be loaded with the value that immediately follows; in this case 20 (hexadecimal). Immediate addressing is very fast since the value to be loaded is included in the instruction. However, since the value to be loaded is fixed at compile-time it is not very flexible.

**Direct Addressing**

Direct addressing is so-named because the value to be stored in memory is obtained by directly retrieving it from another memory location. For example:
\[\text{MOV A,30h}\]

This instruction will read the data out of Internal RAM address 30 (hexadecimal) and store it in the Accumulator. Direct addressing is generally fast since, although the value to be loaded is not included in the instruction, it is quickly accessible since it is stored in the 8051s Internal RAM. It is also much more flexible than Immediate Addressing since the value to be loaded is whatever is found at the given address—which may be variable.

Also, it is important to note that when using direct addressing any instruction which refers to an address between 00h and 7Fh is referring to Internal Memory. Any instruction which refers to an address between 80h and FFh is referring to the SFR control registers that control the 8051 microcontroller itself.

The obvious question that may arise is, "If direct addressing an address from 80h through FFh refers to SFRs, how can I access the upper 128 bytes of Internal RAM that are available on the 8052?" The answer is: You cant access them using direct addressing. As stated, if you directly refer to an address of 80h through FFh you will be referring to an SFR. However, you may access the 8052s upper 128 bytes of RAM by using the next addressing mode, "indirect addressing."
Indirect Addressing

Indirect addressing is a very powerful addressing mode which in many cases provides an exceptional level of flexibility. Indirect addressing is also the only way to access the extra 128 bytes of Internal RAM found on an 8052.

Indirect addressing appears as follows:

\[ MOV A, @R0 \]

This instruction causes the 8051 to analyze the value of the R0 register. The 8051 will then load the accumulator with the value from Internal RAM which is found at the address indicated by R0.

For example, let's say R0 holds the value 40h and Internal RAM address 40h holds the value 67h. When the above instruction is executed the 8051 will check the value of R0. Since R0 holds 40h the 8051 will get the value out of Internal RAM address 40h (which holds 67h) and store it in the Accumulator. Thus, the Accumulator ends up holding 67h.

Indirect addressing always refers to Internal RAM; it never refers to an SFR. Thus, in a prior example we mentioned that SFR 99h can be used to write a value to the serial port. Thus one may think that the following would be a valid solution to write the value 1 to the serial port:

\[ MOV R0, #99h \quad ; Load the address of the serial port \]
\[ MOV @R0, #01h \quad ; Send 01 to the serial port -- WRONG!! \]

This is not valid. Since indirect addressing always refers to Internal RAM these two instructions would write the value 01h to Internal RAM address 99h on an 8052. On an 8051 these two instructions would produce an undefined result since the 8051 only has 128 bytes of Internal RAM.

External Direct

External Memory is accessed using a suite of instructions which use what I call "External Direct" addressing. I call it this because it appears to be direct addressing, but it is used to access external memory rather than internal memory.

There are only two commands that use External Direct addressing mode:

\[ MOVX A, @DPTR \]
\[ MOVX @DPTR, A \]

As you can see, both commands utilize DPTR. In these instructions, DPTR must first be loaded with the address of external memory that you wish to read or write. Once DPTR holds the correct external memory address, the first command will move the contents of that external memory
address into the Accumulator. The second command will do the opposite: it will allow you to write the value of the Accumulator to the external memory address pointed to by DPTR.

**External Indirect**

External memory can also be accessed using a form of indirect addressing which I call External Indirect addressing. This form of addressing is usually only used in relatively small projects that have a very small amount of external RAM. An example of this addressing mode is:

\[ \text{MOVX @R0, A} \]

Once again, the value of R0 is first read and the value of the Accumulator is written to that address in External RAM. Since the value of @R0 can only be 00h through FFh the project would effectively be limited to 256 bytes of External RAM. There are relatively simple hardware/software tricks that can be implemented to access more than 256 bytes of memory using External Indirect addressing; however, it is usually easier to use External Direct addressing if your project has more than 256 bytes of External RAM.

[1] Write an 8051 assembly language program to load 0FFH the registers A, R2, R3 and R4.

<table>
<thead>
<tr>
<th>Label</th>
<th>Opcode</th>
<th>Operand/s</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORG</td>
<td>0000H;</td>
<td></td>
<td>Store the program codes starting at 0000H</td>
</tr>
<tr>
<td>START</td>
<td>MOV</td>
<td>A, #0FFH;</td>
<td>Load accumulator. (Load 0FFH to accumulator)</td>
</tr>
<tr>
<td></td>
<td>MOV</td>
<td>R2, #0FFH;</td>
<td>Load register R2. (Load 0FFH to R2)</td>
</tr>
<tr>
<td></td>
<td>MOV</td>
<td>R3, #0FFH</td>
<td>Load register R3. (Load 0FFH to R3)</td>
</tr>
<tr>
<td></td>
<td>MOV</td>
<td>R4, #0FFH</td>
<td>Load register R4. (Load 0FFH to R4)</td>
</tr>
<tr>
<td>END</td>
<td></td>
<td></td>
<td>End of the program</td>
</tr>
</tbody>
</table>

Enter the above program using Keil software, save, debug, run and examine the register contents.

☐ Refer to the procedure described in the manual.

**Exercise (To be executed in the Lab)**
[1] WAP to load the register R0, R1, R2, R3 with the value 32 H, E4 H, 8D H, 83 H respectively.
[2] WAP to transfer data from accumulator to external memory address 078E H.
[3] WAP to load 4AH data in code memory at location 0700H.

**Assignment:**

[1] Write an 8051 program to load DPTR register with the data E8F0H, using at least four different methods.

[2] Repeat the demonstration program by using addresses of registers in place of register names. Loading accumulator first with 0FFH and then transferring accumulator content to other registers.

[3] WAP to transfer data from register R1 to external memory address 078E H.

[4] WAP to load set of 4 data (34H, E3H, 56H, 9CH) in code memory, starting from 0700H.

[5] WAP to transfer the content of Code memory address 02A7 H, in to R1 register.

(* mention the register mode in comments of each instruction)
Experiment 3: Programs of Data block transfer

**Aim:** Write assembly language programs to transfer a data block from one place to another place in the memory of Microcontroller 8051.

**Equipment used:** Computer System.

**Software Used:** Keil uVision Software.

**Description:**
Data block transfer mainly involves the counting and comparing of numbers. The instructions associated with this experiment is **DJNZ**.

**DJNZ:** The DJNZ instruction decrements the byte indicated by the first operand and, if the resulting value is not zero, branches to the address specified in the second operand.

**Format:**

(i)  DJNZ direct, offset ; 3 bytes, 2 cycles  

DJNZ  
PC = PC + 2  

Operation:  
(direct) = (direct) - 1  
IF (direct) <> 0  
PC = PC + offset

*Example: DJNZ 40H, Label*

(ii) DJNZ Rn, offset ; 2 bytes, 2 cycles  

DJNZ  
PC = PC + 2  

Operation:  
Rn = Rn - 1  
IF Rn <> 0  
PC = PC + offset

*Example: DJNZ R7, Label*
Exercise (To be executed in the Lab)

[1] WAP to transfer the data block (array) of 8 elements (available at RAM location 30H to 37H) at new location of starting address 50H in sequence.

[2] WAP to transfer the data block (array) of 8 elements (available at RAM location 30H to 37H) at new location of starting address 50H in reverse sequence.

[3] WAP to exchange the data block (array) of 8 elements (available at RAM location 30H to 37H) at new location of starting address 50H in sequence. (Without exchange instruction)

[4] WAP to exchange the data block (array) of 8 elements (available at RAM location 30H to 37H) at new location of starting address 50H in reverse sequence. (Without exchange instruction)

Assignment:

[1] WAP to break the data block (array) of 8 elements (available at RAM location 30H to 37H) in 2 sections, and first section is transferred at new location of starting address 50H in sequence and second section is transferred at new location of starting address 60H in sequence.

[2] WAP to break the data block (array) of 8 elements (available at RAM location 30H to 37H) in 2 sections, and first section is transferred at new location of starting address 50H in reverse sequence and second section is transferred at new location of starting address 60H in sequence.

[3] WAP to transfer the block of data from 20h to 30h to external location 1020h to 1030h.

[4] A set of ‘13’ data bytes are available in successive RAM locations starting at 40H. Write an 8051 program to transfer these data bytes to successive locations starting at 46H.

[5] An array of N’ data bytes are stored in memory locations starting at 61H. The length of the block N is stored in memory location 60H. Write an ALP to reverse the array of bytes in its own place, without using additional memory.
Experiment 4: Programs of Searching of data in array

**Aim:** Write assembly language programs to search a number in block available in memory of Microcontroller 8051.

**Equipment used:** Computer System.

**Software Used:** Keil uVision Software.

**Description:**
The basic concept of searching of a number in a data block, involves the comparing of numbers. In microcontroller 8051 assembly language, CJNE is an essential instructions, compares the first two operands and branches to the specified destination if their values are not equal. If the values are the same, execution continues with the next instruction. It affects the carry flag.

**CJNE:** Compare _ and _ Jump if not Equal

Example:
(i)  CJNE @R1, #24H, LABEL  
(ii) CJNE A, #01H, LABEL  
(iii) CJNE A, 60h, LABEL  
(iv) CJNE R6, #12H, LABEL

**Exercise:**
[1] Write an ALP to find the data 89H is available or not in the array available at ten successive location starting at 40H, load 00H in R0 if the data is not available in the array, otherwise load 01H in R0.
[2] Write an ALP to find the number of repetition occur of data 89H in array available at ten successive location starting at 30H, store the result in the register R1.
[3] Write an ALP to find the maximum number of the array available at ten successive location starting at 30H, store the result in the register R0.

**Assignment:**
[1] Write an ALP to find the minimum number of the array available at ten successive location starting at 40H, store the result in the register R0, also find the number of repetition of the maximum number in given array, and store in register R1.
[2] Write an ALP to find the minimum number of the array of 10 elements (each of 1 byte long) available in the external RAM from starting address 0400H, store the result in the register R0, also find the number of repetition of the maximum number in given array, and store in register R1.

[3] Write an ALP to find the minimum number of the array of 10 elements (each of 2 byte long) available in the external RAM from starting address 0400H, store the result in the register DPTR, also find the number of repetition of the maximum number in given array, and store in register R1.
Experiment 5: Programs of Sorting of Data Array

Aim: Write and perform programs to sort a data block available in memory of Microcontroller 8051.

Equipment used: Computer System.

Software Used: Keil uVision Software.

Description:
Sorting means arranging a set of data in some order. There are different methods that are used to sort the data in ascending & descending order. Some of the methods are:

- Selection sort
- Bubble sort
- Insertion sort

Selection sort
Here, to sort the data in ascending order, the 0th element is compared with all other elements. If it is greater than the compared element then they are interchanged. So after the first iteration smallest element is placed at the 0th position. The same procedure is repeated for the 1st element and so on.

Bubble sort
Here, to sort the data in ascending order, 0th element is compared with the 1st element. If it is greater than the 1st element then they are interchanged else remains same. Then 1st element is compared with 2nd element and procedure is continued. So after the first iteration largest element is placed at the last position. In the second iteration, the comparisons are made till the last but one position and now second largest element is placed at the second last position. After all the iterations, the list becomes sorted.

Insertion sort
Here, sorting takes place by inserting a particular element at the appropriate position. To sort the element in ascending order, in first iteration, 1st element is compared with the 0th element. In the second iteration 2nd element is compared with 0th and 1st element. In general, in every iteration an element is compared with all the elements before it. While comparing if it is found that the element in question can be inserted at a suitable position, then space is created for it by shifting the other element one position to the right and inserting the element at the suitable position. This procedure is repeated for all the elements in the list.
The CJNE instruction & DJNZ instructions can be used for performing this experiments. These instructions are already described in previous experiments.

**Exercise:**

[1] Write an ALP to sort the array in ascending manner, available at ten successive location starting at 40H, store the sorted values in new array of starting address of 50H.

[2] Write an ALP to sort the array in descending manner, available at ten successive location starting at 40H, store the sorted values in new array of starting address of 50H.

**Assignment:**

[1] Write an ALP to sort the array in ascending manner, available at ten successive location starting at 40H, store the sorted values at same location.

[2] Write an ALP to sort the array of 10 elements (each element is 16 bit long) in ascending manner, available at ten successive location starting at 40H, store the sorted values in new array of starting address of 50H.

[3] Write an ALP to sort the array of 10 elements (each element is 8 bit long) in descending manner, available at ten successive location in external RAM of starting address 0340H, store the sorted values in new array of starting address of 50H (internal RAM Address).
Experiment 6: Programs of Arithmetic & Logical Operations – I

**Aim:** Write assembly language programs for mathematical operation (addition and subtraction) using Microcontroller 8051.

**Equipment used:** Computer System

**Software Used:** Keil uVision Software.

**Description:**

The basic instructions for the addition & subtraction in assembly language is **ADD**, **ADDC**, **SUBB**.

**ADD:** The **ADD** instruction adds a byte value to the accumulator and stores the results back in the accumulator. Several of the flag registers are affected.

The all possible way of writing **ADD** instructions are as follow:

1. `ADD A, #immediate`
2. `ADD A, @Ri`
3. `ADD A, direct`
4. `ADD A, Rn`

**ADDC:** The **ADDC** instruction adds a byte value and the value of the carry flag to the accumulator. The results of the addition are stored back in the accumulator. Several of the flag registers are affected.

The all possible way of writing **ADDC** instructions are as follow:

1. `ADDC A, #immediate`
2. `ADDC A, @Ri`
3. `ADDC A, direct`
4. `ADDC A, Rn`

**SUBB:** The **SUBB** instruction subtracts the specified byte variable and the carry flag from the accumulator. The result is stored in the accumulator. This instruction sets the carry flag if a borrow is required for bit 7 of the result. If no borrow is required, the carry flag is cleared.

The all possible way of writing **SUBB** instructions are as follow:

1. `SUBB A, #immediate`
2. `SUBB A, @Ri`
3. `SUBB A, direct`
4. `SUBB A, Rn`

**Exercise:**

[1] Write an 8051 program to add the contents of register R5 to that of register R6. Store the result in RAM location 30H. Verify PSW register content and comment. Use immediate addressing to load the values to R5 and R6.
[2] Write an 8051 program to subtract the contents of register R5 to that of register R6. Store the result in RAM location 30H. Verify PSW register content and comment. Use immediate addressing to load the values to R5 and R6.

[3] A set of one byte unsigned binary numbers is stored from memory location 51H onwards, the size of the set is available at 50H. Write an 8051 program to add these numbers and store the 16-bit result in 70H and 71H.

[4] A set of one byte signed binary numbers is stored from memory location 40H onwards, the size of the set is available at 50H. Write an 8051 program to add these numbers and store the 16-bit result in 60H and 61H.

[5] Write an ALP to find whether the 8 bit binary number stored at external RAM location 1050H is of even parity or odd parity. If parity is odd, store the number in 1051H, otherwise, store in 1052H.

[6] Write an ALP to set bits D3 and D7 of RAM locations 2DH and 3EH without affecting other bits,

✓ Without using bit manipulation instructions and
✓ Using bit manipulation instructions.

➢ Check the output using single step execution.

Assignment:

[1] A set of one byte unsigned binary numbers is stored from memory location 51H onwards, the size of the set is available at 50H. Write an 8051 program to add these numbers and store the 16-bit result in 70H and 71H.

[2] Ten unsigned ‘8’ bit numbers are available in successive RAM locations starting at 45H. Write an ALP to add these numbers and store the 16 bit result in the next two locations. Ignore the numbers greater than 63H while adding the numbers.

[3] Write an ALP to subtract one unsigned 8-bit number at external RAM location 7000H from the other at 7001H. If the result is positive, store 00H in 7002H and the result in 7003H. If the result is negative store 01H in 7002H and the magnitude of the actual difference in 7003H.

[4] Five signed ‘8’ bit numbers are available in RAM locations starting at 55H. Write an ALP to subtract these numbers from another set of five signed ‘8’ bit numbers available in locations starting at 60H. Store the difference in locations starting at 70H.
Experiment 7: Programs of Arithmetic & Logical Operations – II

Aim: Write and perform programs for mathematical operation (multiplication and division) using Microcontroller 8051.

Equipment used: Computer System.

Software Used: Keil uVision Software.

Description:
The MUL & DIV instruction for the multiplication & division of numbers in assembly language programming.

MUL: The MUL instruction multiplies the unsigned 8-bit integer in the accumulator and the unsigned 8-bit integer in the B register producing a 16-bit product. The low-order byte of the product is returned in the accumulator. The high-order byte of the product is returned in the B register. The OV flag is set if the product is greater than 255 (0FFh), otherwise it is cleared. The carry flag is always cleared.

Example: MUL AB

DIV: The DIV instruction divides the unsigned 8-bit integer in the accumulator by the unsigned 8-bit integer in register B. After the division, the quotient is stored in the accumulator and the remainder is stored in the B register. The carry and OV flags are cleared. If the B register begins with a value of 00h the division operation is undefined, the values of the accumulator and B register are undefined after the division, and the OV flag will be set indicating a division-by-zero error.

Example: DIV AB

Exercise:
[1] Write an ALP to multiply two 8 bit unsigned numbers available at internal RAM locations 51H and 52H. Store the 16-bit result in the next two consecutive locations.
[2] Write an ALP to multiply an 8 bit unsigned number at location 60H by another 8 bit signed number at 61H and store the 16 bit result in the next two locations.
[3] WAP to divide the content of R0 by R1. Store the result in R2 (answer) and R3 (reminder). Then restore the original content of R0.
[4] Write an ALP to check whether the 8 bit unsigned number at 65H is divisible by 05H. If divisible store the quotient at 66H. Otherwise store the quotient and remainder at 67H and 68H.
[5] Write an ALP to add two signed numbers in 2’s compliment form available at RAM locations 3DH and 3EH and store the result in 3FH. Check the overflow flag and comment on the result.

Assignment:

[1] Write an ALP to multiply two 8 bit unsigned numbers available at external RAM locations 8051H and 8052H. Store the 16-bit result in the next two consecutive locations.

[2] Write an ALP to multiply an 8 bit signed number at location 60H by another 8 bit signed number at 61H and store the 16 bit result in the next two locations. Use sign and magnitude form for representing the signed numbers.

[3] Write an ALP to multiply an 8 bit signed number at location 60H by another 8 bit signed number at 61H and store the 16 bit result in the next two locations. Use sign and magnitude form for representing the signed numbers.
Experiment 8: Programs of BCD Number Operations

**Aim:** Write and perform programs of mathematical operations on BCD Numbers.

**Equipment used:** Computer System.

**Software Used:** Keil uVision Software.

**Descriptions:**

BCD stands for binary coded decimal. BCD is needed because in everyday life we use the digits 0 to 9 for numbers, not binary or hex numbers. Binary representation of 0 to 9 is called BCD (see Figure 6-1). In computer literature one encounters two terms for BCD numbers, (1) unpacked BCD, and (2) packed BCD.

In unpacked BCD, the lower 4 bits of the number represent the BCD number, and the rest of the bits are 0. For example, “0000 1001” and “0000 0101” are unpacked BCD for 9 and 5, respectively. Unpacked BCD requires 1 byte of memory or an 8-bit register to contain it.

In packed BCD, a single byte has two BCD numbers in it, one in the lower 4 bits, and one in the upper 4 bits. For example, “0101 1001” is packed BCD for 59H. It takes only 1 byte of memory to store the packed BCD operands. And so one reason to use packed BCD is that it is twice as efficient in storing data.

The DA (decimal adjust for addition) instruction in the 8051 is provided to correct the aforementioned problem associated with BCD addition. The mnemonic “DA” has as its only operand the accumulator “A”. The DA instruction will add 6 to the lower nibble or higher nibble if needed; otherwise, it will leave the result alone. The following example will clarify these points.

```
MOV A, #47H ; A=47H first BCD operand
MOV B, #25H ; B=25H second BCD operand
ADD A, B ; hxx(hxixx) addition (A=6CH)
DA A ; adjust for BCD addition (A=73H)
```

**Exercise:**

[1] Write an 8051 program to add ‘n’ 8-bit BCD numbers available in RAM locations starting at 30H. Store the 16 bit BCD result in 41H and 42H. Use assembler directive EQU to indicate the value of ‘n’.

[2] Write an 8051 program to subtract an ‘8’ bit packed BCD number available in the RAM location 33H from another 8’ bit packed BCD number available at 34H. If the result is positive, store 00H in
35H and the BCD result in 36H. If the result is negative store 01H in 35H and the magnitude of the actual difference in 36H.

Assignment:

[1] Write an ALP to add two multi byte BCD numbers. First number starts at RAM location 40H and the second number starts at 50H. Store the BCD result in locations starting at 70H, starting with the byte of the result. Use the assembler directive EQU to indicate the number of bytes of the numbers.

[2] Write an ALP to subtract multibyte-number-1 from multibyte-number-2. First number starts at RAM location 35H and the second number starts at 45H. Store the difference in successive locations starting at 55H. Number of bytes of both the numbers is available at 34H.
Experiment 9: Programs of Code Conversion

Aim: Write and perform programs of the various number system based code conversion operation.

Equipment used: Computer System.

Software Used: Keil uVision Software.

Descriptions:

There are various number system used to represent the numbers in digital domain. Few of them are Binary, Hexadecimal, Octal, Decimal, ASCII, Excess-3, Gray Code etc.

It is often necessary to convert values written in one number system to another. The simplest way is to reach for your calculator or use a conversion app. from the web. That is fine, but converting a number in this way does not help you to understand the way each number system works and how different systems are related. The purpose of this module is to explain just that, and to get you to carry out some simple conversions so that you can not only convert between number systems, but also understand how the conversion process works. There are various ways to tackle conversions without a calculator; once the conversion methods are learned, the only skills needed are the ability to multiply and divide, and to add together a few numbers.

Conversion from any system to decimal

The number of values that can be expressed by a single digit in any number system is called the system radix, and any value can be expressed in terms of its system radix.

➤ Octal to Decimal

For example the system radix of octal is 8, since any of the 8 values from 0 to 7 can be written as a single digit.

Convert 126\text{\_\text{\textit{8}}} to decimal.

Using the values of each column, (which in an octal integer are powers of 8) the octal value 126\text{\_\text{\textit{8}}} can also be written as:

\((1\times8^2) + (2\times8^1) + (6 \times 8^0)\)

As \((8^2 = 64), (8^1 = 8)\) and \((8^0 =1)\), this gives a multiplier value for each column.

Multiply the digit in each column by the column multiplier value for that column to give:
1\times64 = 64 \quad 2\times8 = 16 \quad 6\times1 = 6

Then simply add these results to give the decimal value.

64 + 16 + 6 = 86_{10}

Therefore $126_8 = 86_{10}$.

- **Binary to Decimal**

Convert $1101_2$ to decimal.

The same method can be used to convert binary number to decimal:

$$= (1\times2^3) + (1\times2^2) + (0\times2^1) + (1\times2^0)$$

$$= 8 + 4 + 0 + 1$$

$$= 13_{10}$$

Therefore $1101_2 = 13_{10}$.

- **Hexadecimal to Decimal**

Convert $B2D_{16}$ to decimal.

Using the same method to convert hexadecimal to decimal.

$$= (B\times16^2) + (2\times16^1) + (D\times16^0)$$

$$= (11\times16^2) + (2\times16^1) + (13\times16^0)$$

$$= 2816 + 32 + 13$$

$$= 2861_{10}$$

Therefore $B2D_{16} = 2861_{10}$.

The same method (multiplying each digit by it's column value) can be to convert any system to decimal.
Exercise:
[1] Write an ALP to convert a 2 digit Hex number available in RAM location 6BH to its equivalent BCD number and display it in packed form in the next two RAM locations.
[2] Write an ALP to unpack a packed Hex number between 00 and FF stored at RAM location 50H and convert each of the unpacked number to its equivalent ASCII and display them in port ‘0’ and port ‘1’. Use look up table technique to find the ASCII code. Use assembler directive DB to store the table of ASCII codes.

Assignment:
[1] A set of 5 packed BCD numbers is available in RAM locations starting at 35H. Write an ALP to convert these numbers to equivalent binary and store them in consecutive locations starting at 45H.

[2] Write an ALP to convert an ASCII code available at RAM location 7CH to equivalent Hex number and store it in the next RAM location.

[3] Write an ALP to unpack a packed Hex number between 00 and FF stored at RAM location 50H and convert each of the unpacked number to its equivalent ASCII and display them in port ‘0’ and port ‘1’. Use look up table technique to find the ASCII code. Use assembler directive DB to store the table of ASCII codes.
Experiment 10: Programs of Delay Generation, Timer & Counter

Aim: Write and simulate programs for Delay Generation, Timer & Counter operations.

Equipment used: Computer System.

Software Used: Keil uVision Software.

Descriptions:
The 8051 microcontroller has two independent 16 bit up counting timers named Timer 0 and Timer 1 and this article is about generating time delays using the 8051 timers. Generating delay using pure software loops have been already discussed here but such delays are poor in accuracy and cannot be used in sensitive applications. Delay using timer is the most accurate and surely the best method.

A timer can be generalized as a multi-bit counter which increments/decrements itself on receiving a clock signal and produces an interrupt signal up on roll over. When the counter is running on the processor’s clock, it is called a “Timer”, which counts a predefined number of processor clock pulses and generates a programmable delay. When the counter is running on an external clock source (may be a periodic or aperiodic external signal) it is called a “Counter” itself and it can be used for counting external events.

In 8051, the oscillator output is divided by 12 using a divide by 12 network and then fed to the Timer as the clock signal. That means for an 8051 running at 12MHz, the timer clock input will be 1MHz. That means the the timer advances once in every 1uS and the maximum time delay possible using a single 8051 timer is \( (2^{16}) \times (1\mu S) = 65536\mu S \). Delays longer than this can be implemented by writing up a basic delay program using timer and then looping it for a required number of time. We will see all these in detail in next sections of this article.

Designing a delay program using 8051 timers:

While designing delay programs in 8051, calculating the initial value that has to be loaded into TH and TL registers forms a very important thing. Let us see how it is done.

- Assume the processor is clocked by a 12MHz crystal.
- That means, the timer clock input will be 12MHz/12 = 1MHz
- That means, the time taken for the timer to make one increment = 1/1MHz = 1uS
- For a time delay of “X” uS the timer has to make “X” increments.
- \( 2^{16} = 65536 \) is the maximum number of counts possible for a 16 bit timer.
- Let TH be the value value that has to be loaded to TH register and TL be the value that has to be loaded to TL register.
• Then, \( \text{THTL} = \) Hexadecimal equivalent of \((65536-X)\) where \((65536-X)\) is considered in decimal.

Example. Let the required delay be \(1000\mu\text{S}\) (ie; \(1\text{mS}\)).
That means \(X = 1000\)

\(65536 - X = 65536 - 1000 = 64536.\)
64536 is considered in decimal and converting it to hexadecimal gives FC18

That means \(\text{THTL} = \text{FC18}\)

Therefore \(\text{TH} = \text{FC}\) and \(\text{TL} = 18\)

Few points to remember while using timers:
• Once timer flag (TF) is set, the programmer must clear it before it can be set again.
• The timer does not stop after the timer flag is set. The programmer must clear the TR bit in order to stop the timer.
• Once the timer overflows, the programmer must reload the initial start values to the \(\text{TH}\) and \(\text{TL}\) registers to begin counting up from.
• We can configure the desired timer to create an interrupt when the TF flag is set.
• If interrupt is not used, then we have to check the timer flag (TF) is set using some conditional branching instruction.
• Maximum delay possible using a single 8051 timer is \(65536\mu\text{S}\) and minimum is \(1\mu\text{S}\) provided that you are using a 12MHz crystal for clocking the microcontroller.
Exercise:
[1] Write an ALP for generate a square wave of 10KHz at Port pin P1.1 without timer.
[2] Write an ALP for generate a square wave of 10KHz at Port pin P1.1 with the help of timer.
[3] Write an ALP to implement a decimal DOWN counter from 95 - 30 and display it continuously in the external RAM location 1000H with a delay of 1.1 secs between each count. Use eight bit registers of 8051 to generate delay.
[4] Write an ALP for 4 digit hexadecimal DOWN counter from 2003 to 1F90 with a delay of 0.8secs and display the count in ports 1& 2. Use timer ‘0’ in mode ‘2’ to generate the required delay.

Assignment:
[1] Write an ALP to implement a decimal UP counter from 25 - 69 and display it continuously in the RAM location 33H with a delay of 0.4 secs between each count. Use 8051 instructions to generate delay.

[2] Write an ALP to realize and display a digital clock in hours, minutes and seconds at the RAM locations 44H, 45H and 46H respectively.
MODULE 2

Interfacing Exercises
These exercises are conducted using ESAMCB51 microcontroller trainer kits.

INSTALLATION
- Make sure that the switch SW1 is kept in the Exec Mode before connecting.
- Select self-powered (jumper JP1 in 1 – 2 position) or USB powered mode of power (jumper JP1 in 3 – 4 position)
- Connect the power supply through power jack J1 in case of self-powered mode and through USB cable via USB connector CN1 in case of USB powered mode.
- When powered on the sign on message ESA MCB 51 V1.03 should appear on the LCD.

ESA MCB 51 MONITOR ROUTINES AND LCD LIBRARY FUNCTIONS
ESA MCB 51 Monitor provides the user with several useful monitor and LCD routines that significantly simplifies the task of program development. Some of them are described below.

Name of routine: LCDINIT
Function: Initializes LCD with minimum settings like 5x7 font, No cursor & Blink off, Clear display.
Call Address: 0E267h

Name of routine: LCDCMD
- Function: Writes the Commands to the Instruction register of LCD
- Input: Store the Command byte in accumulator.
- Call Address: 0E295h

Name of routine: LCDDAT
- Function: Writes the ASCII Characters to the Data register of LCD
- Input: Store the ASCII Character in accumulator.
- Call Address: 0E2B4h

* Name of routine: LCDSTATUS
- Function: Reads the Busy Flag of LCD and comes out the routine when Busy Flag becomes low.
- Call Address: 0E2D3h

Name of routine: LCDSTR

Department of Electrical Engineering
Function: Displays string pointing by DPTR register and the string should be terminated with 00h.

Input: Store the starting address of the string in DPTR register and the string should be stored in CODE memory.

Call Address: 0E39Eh

Name of routine: HEX2ASCII

Function: This Routine converts the lower nibble of accumulator to ASCII number.

Input: Store the number to be converted in lower nibble of Accumulator.

Call Address: 0E3B0h

Integration with Keil

Start the μVision4 IDE, create a project file with Project – New Project, and select the CPU from the device database. Select AT89C51ED2.

Create and save a source file and add this source file to the project.

Configure flash tools with the following:

Target page: Set the XTAL frequency as 11.0592 MHz, Select the use on-chip ROM option.

Debug page: Select Use Keil Monitor-51 Driver, enable Load Application at Startup Option;

Select appropriate communication settings using the Settings button in the Debug page

Under the COM Port settings select the COM Port to which the board is connected and select the baud rate as 9600.

Enable Serial Interrupt option if the user application is not using on-chip UART, to stop program execution.

Build the project; μVision translates all the user application and links. After successful Build, press Reset on ESA MCB 51.

Now user can enter into Debug mode with Debug- Start / Stop Debug session dialog.

The debugger connects to ESA MCB 51 and downloads the user application program into flash memory and also the debugger enables all the debugging features like Run, Step, Step over etc.

Select Run to execute your program and verify the outputs.

To stop the execution of user application programs, use Stop option in Debug dialog. If the user enabled the Serial Interrupt in Target – Debug settings then the program execution stops normally.
Experiment 11: Simulation of Interrupts Programming

**Aim:** Write and simulate programs for interrupt operation on ESA51 Microcontroller Kit.

**Equipment used:** Computer System, ESA MCB51

**Software Used:** Keil uVision Software.

**Description:**

**INTRODUCTION TO INTERRUPTS**

The Interrupts are Service Routines similar to Subroutines, meant for interrupting the main program, execute the ISR and return back to main program. 8051 microcontroller is provided with six interrupts.

* External Interrupts: INT0 and INT1
* Timer Interrupts: TF0 and TF1
* Serial Communication Interrupt: RI/TI

**RESET**

When 8051 is interrupted via any of the above six interrupts the program control is automatically transferred to a specific location on page ‘00’ in the user program area as shown in the table below.

<table>
<thead>
<tr>
<th>No.</th>
<th>Interrupt</th>
<th>Rom Locaion (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RESET</td>
<td>0000</td>
</tr>
<tr>
<td>2</td>
<td>INT 0</td>
<td>0003</td>
</tr>
<tr>
<td>3</td>
<td>TF 0</td>
<td>000B</td>
</tr>
<tr>
<td>4</td>
<td>INT 1</td>
<td>0013</td>
</tr>
<tr>
<td>5</td>
<td>TF 1</td>
<td>001B</td>
</tr>
<tr>
<td>6</td>
<td>RI / TI</td>
<td>0023</td>
</tr>
</tbody>
</table>

Both the external interrupts are available to the user through keys INT0 and INT1 available on the ESAMCB51 board. From the table it can be seen that only few bytes of memory space are available in the user program memory for all available interrupts. Thus normally LJMP instruction is used as the first instruction to transfer the control to a location where the user writes the required ISR. Upon RESET all the interrupts of 8051 are disabled. Hence 8051 will not respond to any of these interrupts unless they are enabled. The required interrupt can be enabled by setting appropriate bits in the IE (Interrupt Enable) register.
Exercise:

[1] Write a Main program to operate timer ‘0’ in mode 1, enable timer ‘0’ interrupt and display ‘2’ digit decimal up counter from 06H to 44H at port ‘0’ continuously with a delay of 2.5msecs. When timer ‘0’ interrupts the CPU after a delay of 30msecs, write an ISR to get a copy of the present count to port ‘1’ and display ‘2’ digit hexadecimal down counter from this count with a delay of 0.7 secs at port ‘1’. When the count reaches a value of 01, return back to the main program and continue doing it.

[2] Write a main program to enable external interrupts’0’ and ‘1’, and keep waiting for the interrupt. When 8051 is interrupted through INT0, write an ISR to obtain a rectangular wave of 1.5 KHz with 35% duty cycle at P1.6 pin of 8051 continuously and when interrupted through INT1, write an ISR to obtain a rectangular wave of 600Hz with 70% duty cycle at P1.6 pin. Use ‘8’ bit registers of 8051 to obtain the delay.
Experiment 12: Simulation of LCD interface

**Aim:** Write and perform programs to display the data on LCD Screen interfaced with microcontroller 8051.

**Equipment used:** Computer System, ESA MCB51

**Software Used:** Keil uVision Software.

**Description:**
ESAMCB51 contains on board 16 X 2 LCD display unit. LCD modules come in variety of formats like 8x1(eight characters per line and one line), 8x2(eight characters per line and two lines), 16x1(sixteen characters per line and one line), 16x2(sixteen characters per line and two lines), 20x2(twenty characters per line and two lines), 20x4 (twenty characters per line and four lines) etc. LCD accepts characters in ASCII format. Character display font in LCD module is by matrix of dots i.e. each character in LCD module can be represented by 7x5 matrix (7 rows and 5 columns of dots) or 10x5 matrix (10 rows and 5 columns of dots).This interface is built over 16x2 LCD module.

The following table shows the interfacing details of the LCD to MCB51.

<table>
<thead>
<tr>
<th>Pin</th>
<th>Symbol</th>
<th>Function</th>
<th>Port on MCU</th>
</tr>
</thead>
</table>
| 4   | RS     | Register Select  
RS= 0; Instruction register  
RS=1; Data register | P3.7 |
| 5   | R/W    | Read/ Write  
1 = Read, 0 = Write | P3.6 |
| 6   | E      | Enable     | P3.5 |
| 07-14 | DB0- DB7 | Bi-directional data bus | P2.0 - P2.7 |

**Exercise:**

[1] Write an ALP to display the message “WELCOME TO MANIPAL” on the on board LCD display unit. Display “WELCOME” at the centre of the first line and “TO MANIPAL” at the centre of the second line. Do not use any standard LCD routines provided by ESA.
Experiment 13: Simulation of Logic controller Interface

**Aim:** Write and perform programs of various logic operations on ESA51 Microcontroller Kit.

**Equipment used:** Computer System, ESA MCB51, IF-LOGIC.

**Software Used:** Keil uVision Software.

**Description:**

**INTERFACING LOGIC CONTROLLER WITH 8051**

Logic controller interface consists of ‘8’ TTL inputs and ‘8’ TTL outputs. There are eight input pins provided on each card. Each of these can be set to logic 1/logic 0 by using the DIP switch. There are two sets of LEDs indicating the status of input and output ports. The status of inputs can be read through port ‘1’ of 8051 and the status of output lines can be controlled through port ‘0’ of 8051.

**Connections**

* Connect the 26-pin connector on the logic controller card to J3 connector on ESAMCB-51 kit using 26 core flat cable.

* The required Power supply +5V, GND for the interface is drawn from the trainer. (Supply the required power from the power supply to the power jack on the kit)

* Keep all the switches at SW1 on the interface in off position and control the input through switches at SW2.
Exercise:

[1] Interface Logic Controller Interface to 8051 to perform the following logical operations.

✓ Write an ALP to read the input status, compliment bits 3 and 5 without affecting the other bits and display it through output LEDs.

✓ Write an 8051 program to blink the LEDs at the output port for 0.2secs one after the other starting with the LED at the LS bit.

✓ Write an 8051 ALP to perform the following logical operations. Take the input from the input port and display the output at the output port.

(a) \( Y_0 = X_0 \text{ XOR} X_1 \)  
(b) \( Y_2 = (X_1 + X_2) \cdot X_3 \)  
(c) \( Y_4 = (X_0 + X_1) + (X_2 \cdot X_3) \)

[2] Write an 8051 ALP to display decimal down counter from 90H to 05H at the output port of logic controller interface with a delay of ‘1’ sec between each count. Use timer ‘1’ in mode 1 to obtain the delay.
Experiment 14: Simulation of Stepper Motor Interface

**Aim:** Write assembly language programs to interface & run a stepper motor at various speed using ESA51 Microcontroller Kit.

**Equipment used:** Computer System, ESA MCB51, IF-STEPPER, Stepper Motor, Power supply (+5V@250mA, +12V@250mA, +5V@3A).

**Software Used:** Keil uVision Software.

**Descriptions:**

Stepper motors take in electrical signals and output mechanical angular movement. Their high accuracy and low cost make them popular in applications ranging from printers to cameras. Current in the stator coil creates magnetic fields that move a rotor. As the current into the stator is modified the rotor spins at different speeds.

The stepper motor has 6 pins: +5V supply pin, Ground pins, A, B, C, D (4 Control Pins)

The control pins accepts the sequences of digital pulses, and motor moving with fixed angle movement (step size) for a sequence. The sequences also control the direction of rotation as shown in the table below:

```
<table>
<thead>
<tr>
<th>Step</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

The four leads of the stator winding are controlled by four bits of the 8051 port (in this case P1.0 - P1.3). However, since the 8051 lacks sufficient current to drive the stepper motor windings, we must use a driver such as the ULN2003 to energize the stator. Instead of the ULN2003, we are using power transistors as driver. However, notice that if transistors are used as drivers, we must also use diodes to take care of inductive current generated when the coil is turned off. One reason that the ULN2003 is preferable to the use of transistors as drivers is the fact that the ULN2003 has an internal diode to take care of back EMF.

**RR** (Rotate Right) & **RL** (Rotate Left) are used to generate the needful sequence. The time delay between the sequences are introduced to change the speed of rotation.
Exercise:

[1] Write an ALP to run the stepper motor in clock-wise direction with the speed of
   ✓ 2 revolution per second
   ✓ 10 revolution per second
   ✓ 50 revolution per second
   ✓ 100 revolution per second

[2] Write an ALP to run the stepper motor in anti-clock-wise direction with the speed of
   ✓ 2 revolution per second
   ✓ 10 revolution per second
   ✓ 50 revolution per second
   ✓ 100 revolution per second