LABORATORY MANUAL

MICROCONTROLLER LAB (ECL204)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

COLLEGE OF ENGINEERING

THIRUVANANTHAPURAM

2019

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

COLLEGE OF ENGINEERING

THIRUVANANTHAPURAM

 This manual is prepared as per the 2019 ktu B Tech ProgrammeScheme.

AEL333 EMBEDDED SYSTEMS LAB syllabus

Syllabus : L-T-P: 0-0-3 YEAR OF INTRODUCTION - 2019 Credits -2

Course Outcomes: After the completion of the course the student will be able to

CO 1 : Write an Assembly language program/Embedded C program for performing data manipulation.

CO 2 : Develop ALP/Embedded C Programs to interface microcontroller with peripherals

CO 3 : Perform programming/interfacing experiments with IDE for modern microcontrollers..

List of Experiments:

PART –A (At least 6 experiments are mandatory)

These experiments shall be performed using 8051 trainer kit. The programs shall be written either in embedded C or in assembly language.

1. Data transfer/exchange between specified memory locations.
2. Largest and smallest from a series.
3. Sorting (Ascending/Descending) of data.
4. a. Addition / Subtraction of 8 bit data.
   b.Addition / Subtraction of 16 bit data.
   c. Multiplication / Division of 8 bit data.
   d Multiplication / Division of 16 bit data.
5. Sum of a series of 8 bit data.
6. Multiplication by shift and add method.
7. Square / cube / square root of 8 bit data.
8. Matrix addition.
9. LCM and HCF of two 8 bit numbers.
10. Code conversion – Hex to Decimal/ASCII to Decimal and vice versa.

PART –B (At least 4 experiments are mandatory)

Interfacing experiments shall be done using modern microcontrollers such as 8051

or ARM. The interfacing modules may be developed using Embedded C.

1. Time delay generation and relay interface.
2. Display (LED/Seven segments/LCD) and keyboard interface.
3. ADC interface.
4. DAC interface with wave form generation.
5. Stepper motor and DC motor interface.
6. Realization of Boolean expression through port.

Experiment 1
INTRODUCTION TO 8051

Microcontroller is a programmable logic device that has computing and decision making capability similar to that of a CPU of a computer.

The Microcontroller communicates and operates in the binary numbers 0 and 1 called bits. Each Microcontroller has a fixed set of instruction in the form of binary patterns called machine language. However it is difficult for human to communicate in the language of 0s and 1s. Therefore, the binary instructions given abbreviated names called mnemonics, which form the assembly language for given micro controller. An assembler is used to convert assembly language to machine language. For example if we have to add two numbers in A and B. we can use the instruction ADDA,B . This add instruction is an example of mnemonics. Its machine language form will be 58, 65. This 58, 65 can be obtained from microcontroller manual .58 in hexadecimal represents the machine language instruction for ADD 65 represents A, B.

Each microcontroller recognizes and process a group of bits called the word and microcontrollers are classified according to their word length. For example, a controller with an 8 bit word is known as an 8 bit micro controller and a controller with 32 bit word is known as a 32 bit microcontroller

Organization of a Miccontroller Based system

CPU	RAM	ROM
I/O	Timer	Serial Com port

Figure shows the block diagram of a general purpose micro controller system. Micro controller is a self contained system or self sufficient system having CPU, internal RAM, internal ROM, Timers and counters, I/O ports, serial comp port

Micro controller is a specific purpose digital controller that is meant to read data, perform limited calculations on that data and control its environment based on those calculations

APPLICATIONS

1. Measuring instruments such as the oscilloscope, multi meter and the spectrum analyzer

2. Music related equipment such as synthesizers

3. House hold items, such as the microwave oven, door bell, washing machine and television.

4. Defence equipment such as fighter planes missiles and radar.

5. Medical equipment such as blood pressure monitors, blood analyzers and monitoring system

ARCHITECTURE

The accumulator register 'A' :- The most important data register is the A register which acts as the accumulator. It is a mandatory that the A register carry one of the operands for all arithmetic instructions. The other operand may be in memory (RAM) or in any other register.

Register B:- The register B is not a frequently used register, because it can be used as an operand only for some specific operations like multiplication of two numbers, one operand should be in A , and the other should be B. Same is the case for division. But it can store data.

Internal RAM:- Totally, the 8051 has 256 bytes of RAM, but half of it is reserved to act as the "special function registers", that is , the registers which are used to handle the activities of the peripherals of the device. The remaining 128 bytes is what is referred to as internal RAM, and is divided into parts. The first 32 bytes act as register banks 0 to 3; each bank contains 8 data registers named RO to R7. These registers are used for data manipulations and data movement. At a time, only one of these banks is operational. It is possible to switch from the current bank to another bank by using two bits of the PSW. By default, it is bank O that is the current bank. RAM locations from 0 to 7 are set aside for bank 0 ,where R0 is RAM location O, R1 is RAM location1, R2 is location 2, and so on, until memory location 7, which belongs to R7 of bank 0. The second bank of registers R0- R7 starts at RAM location 08H and goes to location of 0F H. The third bank of R0-R7 starts at memory location 1OH and goes to location 17H. Finally RAM locations 18H to IFH are set aside for the fourth bank of R0-R7.

Bank 1 uses the same RAM as the stack A total of 16 bytes from locations 20 H to 2 FH are set aside for bit addressable read/write memory. A total of 80 bytes from locations 30 H to 7FH are used for

read and write storage or what is normally called a scratch pad. These 80 locations of RAM are widely used for the purpose of storing data and parameters by 8051 programmers

Default register bank – Bank O

How to switch register banks? Register bank O is the default when the 8051 is powered up. We can switch to other banks by use of the PSW (program status word) register. Bits D4 and D3 of the PSW are used to select the desired register bank as shown in Table.

	RS1 (psw.4)	RSO (psw.3)
Bank 0	0	0
Bank 1	0	1
Bank 2	1	0
Bank 3	1	1

The D3 and D4 bits of register program status word(psw) are often referred to as psw.4 and psw 3 since they can be accessed by the bit addressable instructions SETB and CLR. For example, “SETB psw 3” will make psw 3 = 1 and select bank register 1

Stack in the 8051:- The stack is a section of RAM used by the CPU to store information temporarily. This information could be data or address. The CPU needs this storage area since there are only a limited number of registers.

How stacks are accessed in the 8051 :- The register used to access the stack is called the SP (stack pointer) register. The stack pointer in the 8051 is only 8 bits wide, which means that RAM location 08 is the first location used the stack by the 8051. The storing of a CPU register in the stack is called a PUSH, and pulling the contents off the stack back into a CPU register is called a pop. In other words, a register is pushed onto the stack to save it and popped off the stack to retrieve it.

Pushing onto the stack: - In the 8051 the stack pointer (sp) points to the last location of the stack. As we push data onto the stack, the stack pointer (sp) is incremented by one. For every byte of data saved on the stack, sp is incremented only once.

Popping from the stack:- Popping the content of the stack back into a given register is the opposite process of pushing .With every pop, the top byte of the stack is copied to the register specified by the instructions and the stack pointer is decremented once

The upper limit of the stack: Locations 08 to OF in the 8051 RAM can be used for the stack. This is because locations 20- 2FH of RAM are reserved for bit addressable memory and must not be used by the stack. If in a given program we need more area, we can change the SP to point to RAM locations 30-7 FH. This is done with the instruction “MOVSP, XX”.

CALL instruction and the stack: In addition using the stack to save registers, the CPU also used the stack to save the address of the instruction just below the CALL instruction. This is how the CPU knows where to resume when it returns from the called subroutine

PSW (program status word) register:- The PSW register is an 8-bit register. It is also referred to as the flag register. Although the PSW register is 8 bits wide, only 6 bits of it are used by the 8051. The two unused bits are user-definable flags. Four of the Flags are called conditional flags, meaning that they indicate some conditions that result after am instruction being executed. These four are CY (carry) AC (auxiliary carry) P (parity) and OV cover flow. The bits psw 3 and psw4 are designated as RSO and RSI, respectively and are used to change the bank registers. The psw5 and psw1 bits are general – purpose status flag bits and can be used by the programmer for any purpose

CY	AC	FO	RSI	RSO	OV	-	P

CY psw 7 carry flag

AC psw 6 Auxiliary carryflag

FO psw 5 Available to the user for general purpose

RSI psw 4 Register Bank selector bit 1

RSO psw 3 Register Bank selector bit 0

OV psw 1 user definable bit

P psw 0 parity flag

RSI	RSO	Register Bank
0	0	0
0	1	1

1 0 2

1 1 3

CY the carry Flag: - this flag is set whenever there is a carry out from the D7 bit. This flag bit is affected after an 8-bit addition or subtraction. It can also be set to 1 or 0 directly by an instruction such as “SETB C” and CLR C” where “SETB C” stands for “set bit carry” and “CLRC” for “clear carry”

Eg.:- MOV A, #9CH

ADD A, # 64 H

CY=1

AC, the auxiliary carry flag

If there is a carry from D3 to D4 during an ADD or SUB operation, this bit is set; otherwise, it is cleared. This flag is used by instructions that perform BCD arithmetic

Eg. MOV A, #9cH

ADDA, # 64 H’

AC=1

P, the parity flag

The parity flag reflects the number of 1s in the accumulator register only. If the A register contains an odd number of Is, then p=1. Therefor, p= 0 if A has an even number of 1s

Eg. MOV A, #9CH

ADD A, # 64H

P=0

OV the overflow flag

This flag is set whenever the result of a signed number operation is too large causing the high – order bit to overflow into the sign bit. In general, the carry flag is used to detect errors in unsigned arithmetic operations. The overflow flag is only used to detect errors in signed arithmetic operations.

ROM

ROM can be 4k on chip and 60 k external ROM or 64k

Addressing modes

The CPU can access data in various ways. The data could be in a register, or in memory, or be provided as an immediate value. These various ways of accessing data are called addressing modes. The various addressing modes of a microprocessor are determined when it is designed, and therefore cannot be changed by the programmer. The 8051 provides a total of five distinct addressing modes. They are as follows.

1. Immediate
2. Register
3. Direct
4. Register Indirect
5. Indexed

1.**Immediate, addressing mode**:- In this addressing mode, the source operand is a constant. In immediate addressing mode, as the name implies, when the instruction is assembled, the operand comes immediately after the opcode. The immediate data must be preceded by the pound sign, "#" This addressing mode can be used to load information into any of the registers including the DPTR register. Examples follows

```
MOV A, #25H          ;load 25H into A
MOV R4, #62          ;    load 62 into R4
MOV DPTR, #4521      ;    DPTR = 4521
```

2.**Register addressing mode** : Register addressing mode involves the use of registers to hold the data to be manipulated.

Eg : MOVA, R0; copy the contents of R0 into A.

The source and destination registers must match in size. In other words, coding “MOV DPTR, A” will give an error, since the source is an 8 bit register and the destination C5 a 16 bit register.

We can move data between the accumulator and Rn (hr n = 0 to 7) but movement of data between Rn register is not allowed. For example, the instruction “MOV R4, R7” is invalid.

3.**Direct addressing modes** : There are 128 bytes of RAM in the 8051. The RAM has been assigned addresses 00 to 7FH

1. RAM locations 00-1FH are assigned to the register banks and stack.

2. RAM locations 20-2FH are set aside as bit addressable space to save single bit data.

3. RAM locations 30-7FH is available as place to save byte sized data.

Although the entire 128 bytes of RAM can be accessed using direct addressing mode, it is most often used to access RAM locations 30-7FH. This is due to the Fact that register bank locations are accessed by the register names R0-R7, but there is no such name for other RAM locations. In the direct addressing mode the data is in RAM memory locations whose address is known, and this address is given as a part of the instruction. Contrast this with immediate addressing mode, in which the operand itself is provided with the instruction. The “#” sign distinguishes between the two modes.

MOV R0, 40H; save content of RAM location 40H in R0 RAM locations. These registers can be accessed in two ways

MOV A, 4 ; is same as

MOV A, R4 ; which means copy R4 into A

4.**Register indirect addressing mode**

In the register indirect addressing mode, a register is used as pointer to the data. Register R0 and R1 are used for this purpose. In other words R2-R7 cannot be used to hold the address of an operand located in RAM when using this addressing mode when R0 and R1 are used as pointers, that is, when they hold the addresses of RAM locations, they must be preceded by the “@” sign, as show below MOV A, @R0; move contents of RAM location whose address is held by R0 into A.

MOV @ R1, B ; move contents of B into RAM locations

whose address is held by R1.

Adv : - one of the advantages of register indirect addressing mode is that it makes accessing data dynamic rather than static as in the case of direct addressing mode. Example shows two cases of copying 55H into RAM locations 40H to 45H.

In solution (b) that there are two instructions that are repeated numerous times. We can create a loop with those two instructions as shown in solution (c) is the most efficient and is possible only because of register indirect addressing mode. Looping is not possible in direct addressing mode. This is the main difference between the direct and register indirect addressing modes.

5.**Indexed addressing modes** is widely used in accessing data elements of look-up table entries located in the program ROM space of the 8051. The instruction used for this purpose is "MOVC A, @ A+DPTR". The 16-bit register DPTR and register A are used to form the address of the data element stores in on-chip ROM. Because the data elements are stored in the program (code) space ROM of the 8051, the instruction MOVC is used instead of MOV. The "c" means code. In this instruction the contents of A are added to the 16bit register DPTR to form the 16 bit address of the needed data.

PORTS

For input output operations 8051 has 4 ports.

PORT 0

Port 0 provides both address and data. The 8051 multiplexes address and data through port 0 to save pins. When ALE=0, it provides data D0-D7, but when ALE = 1 it has address A0-A7. Therefore, ALE is used for de multiplexing address and data with the help of a 74L5373 latch. The pins of PO must be connected externally to a 10k pull-up resistor. This is due to the fact that PO is an open drain, unlike P1, P2 and P3 with external pull-up resistors connected to P0 it can be used as simple Input Out put port, just like P1 and P2. In contrast to port 0, ports P1, P2, and P3 do not need any pull up resistors since they already have pull-up resistors internally.

PORT1 and PORT2

In 8051 based systems with no external memory connection, both P1 and P2 are used as simple Input –Output. However, in 8031/8051 based systems with external memory connections, port 2 must be used along with P0 to provide the 16-bit address for the external memory

PORT3

Occupies a total of 8 pins. It can be be used as input or output. P3 does not need any pull-up resistors. Although port is configured as an input port upon reset, this is not the way it is most commonly used. Ports has the additional function of providing some extremely important signals such as interrupts.

P3 bit	Function	Pin
P3.0	RxD	10
P3.1	TxD	11
P3.2	INTO	12
P3.3	INTI	13
P3.4	T0	14

P3.5	T1	15
P3.6	WR	16
P3.7	RD	17

P3.1 are used for the RXD and TXD serial communications signals. Bits P3.2 and P3.3 are set aside for external interrupts. Bits P3.4 and P3.5 are used for Timers 0 and 1. P3.6 and P3.7 are used to provide the WR and RD signals of external memory connections.

Experiment 2 .

DATA TRANSFER / EXCHANGE BETWEEN SPECIFIED MEMORY LOCATIONS.

AIM

Write a program to transfer data between memory locations using 8051 .

APPARATUS REQUIRED

8051 Microcontroller kit, (0-5V) DC Power Supply

THEORY

1. The data is transferred between two memory locations which are done in blocks.
2. The XCH instruction loads the accumulator with the byte value of the specified operand while simultaneously storing the previous contents of the accumulator in the specified operand.

ALGORITHM

Step 1: Count from memory location is moved to register
Step 2: zero is moved to register
Step 3: mov dptr with the starting address of array
Step 4: content of array location is moved to accumulator
Step 5: Increment dptr
Step 6: dptr addresss is stored in to the stack
Step 7: mov dptr with memory location.ie starting address of destination
Step 8: store the content of accumulator in to register
Step 9: move the value of register into a
Step 10: move the value of a into dpl
Step 11:move the value of a into address stored in dptr
Step 12: increment register
Step 13.pop dptr value from stack
Step 14: decrement register and jump to step 4 if register is non zero
Step 15: Halt the program

Result

Experiment no.3.
FINDING LARGEST AND SMALLEST FROM A SERIES

AIM

Write a program to find smallest and largest number from a series using 8051

APPARATUS REQUIRED

8051 Microcontroller kit, (0-5V) DC Power Supply

THEORY

1. Let Internal memory location (say 40H) has the biggest number i.e. zero.
2. Now the biggest number in internal memory location is stored in memory as the Result.
3. Now compare the first number with internal memory location. If it is greater, move it to internal memory

Algorithm

Step 1: Number of elements in an array is moved from memory location to A
Step 2: Number of elements is moved to register
Step 3: Move 00 to B
Step 4: Increment DPTR to get the first element
Step 5: Element from memory location is moved to accumulator
Step 6: Jump to step 7 if A and B are not equal
Step 7: If carry jump to step 9 else jump to step 8
Step 8: Large number from A is moved to B
Step 9: Increment DPTR to get next number
Step 10: Decrement register and if jump to step 5 if register is non zero else jump to step11
Ste 11: Move large number from B to A
Step 12: Largest number is moved to the memory location
Step 13: Halt the program

Result

Experiment 4
SORTING IN ASCENDING/DESCENDING ORDER

AIM

Write a program to sort numbers in ascending and descending order using 8051

APPARATUS REQUIRED

8051 Microcontroller kit, (0-5V) DC Power Supply

THEORY

ASCENDING ORDER

1. The sorting technique used here is relatively simple.

2. First consider the first two numbers of the array.

3. Sort according to which is from lowest to highest.

DESCENDING ORDER

1. The sorting technique used here is relatively simple.

2. First consider the first two numbers of the array.

3. Sort according to which is from highest to lowest.

ALGORITHM:

STEP 1: Load accumulator with no. of elements from memory location

STEP2: Decrement no. of elements to obtain no. of steps in first cycle.

Step3: move the first value to accumulator from 4301 and then move to Register

Step4 : move the value of no. of searches to R5

Step 5: move the second value to accumulator from next memory location.

Step 6: move this value to B

Step7: move the content of Register to accumulator.

Step8: compare the two no's and if not in decreasing order proceed to step 10 else move to step 11,

else proceed to next step

Step 9: if carry is found on comparing the two no's ie,the no's are in descending order swap the no's

Step 10: decrement R5 and if not zero proceed to step 5

Step 11: decrement R4 and if not zero move to step 3.

Step 12: halt the program.

Result

Experiment 5

BASIC ARITHMETIC AND LOGIC OPERATIONS

a)Addition of two 8 bit numbers

Algorithm

Step 1: First number is moved from memory location to accumulator
Step 2: Increment DPTR in order to point second number
Step 3: First number is moved to Register
Step 4: Second number is moved to accumulator
Step 5: Add first number and second number
Step 6: Increment DPTR in order to point the result
Step 7: Result is stored in memory location
Step 8: Increment DPTR to point carry
Step 9: Clear accumulator
Step 10: If carry is zero ,then follow step12; otherwise step 10
Step 11: Add 01 to the accumulator in order to represent the carry
Step 12: Move carry status to memory location
Step 13: Halt the program

b)Subtraction two 8 bit numbers

Algorithm

Step 1:First number is moved from memory location to accumulator
Step 2: Increment DPTR in order to point second number
Step 3: First number is moved to register
Step 4: Second number is moved to accumulator
Step 5: Subtract first and second number
Step 6: Increment DPTR in order to point result
Step 7: Result is stored in memory
Step 8: Halt the program

c)Multiplication of two 8 bit numbers

Algorithm

Step 1: First number is loaded from memory location to accumulator
Step2: Increment DPTR pointing second number
Step 3: First number is moved to B
Step 4: Second number is moved to accumulator
Step 5: Multiply first number and second number
Step 6: Increment DPTR in order to store the lowest 8 bit result

Step 7: The lowest 8 bit is stored in DPTR location
Step 8: Increment DPTR in order to store upper 8 bits
Step 9: Content of B is moved to A
Step 10: Store upper 8 bits in the memory location
Step 11: Halt the program
Result

d)Division of 8 bit numbers

Algorithm
Step 1: First number is loaded from memory location to accumulator
Step 2: Increment DPTR pointing the second number
Step 3: First number is moved to B
Step 4: Second number is moved to accumulator
Step 5: Divide First number by second number
Step 6: Increment DPTR in order to store the quotient
Step 7: Quotient is stored in the DPTR location
Step 8: Increment DPTR
Step 9: Content of B is moved to accumulator
Step 10: Store the reminder in the memory location
Step 11: Halt the program

e)AND operation of two 8 bit numbers

Algorithm
Step 1: First number is loaded from memory location to the accumulator
Step 2: Increment DPTR pointing second number
Step 3: First number is moved to B
Step 4: Second number is moved to A
Step 5: AND operation of first and second number
Step 6: Increment DPTR to store the result
Step 7: Result is stored in DPTR location
Step 8: Halt the program

Result

f)OR operation of two 8 bit number

Algorithm
Step 1: First number is loaded from memory location to the accumulator
Step 2: Increment DPTR pointing second number

Step 3: First number is moved to B
Step 4: Second number is moved to A
Step 5: OR operation of first and second number
Step 6: Increment DPTR to store the result
Step 7: Result is stored in DPTR location
Step 8: Halt the program

Result

g) EXOR operation of two 8 bit numbers

Algorithm

Step 1: First number is loaded from memory location to the accumulator
Step 2: Increment DPTR pointing second number
Step 3: First number is moved to B
Step 4: Second number is moved to A
Step 5: EX-OR operation of first and second number
Step 6: Increment DPTR to store the result
Step 7: Result is stored in DPTR location
Step 8: Halt the program

Result

Experiment .6

SUM OF A SERIES OF 8 BIT DATA

AIM

Write a program to find the sum of series of first n 8 bit natural numbers using 8051.

APPARATUS REQUIRED

8051 Microcontrollerr kit, (0-5V) DC Power Supply

THEORY

1. Sum of n natural numbers can be found out by the equation n(n+1)/2.

2. Here it is found out by decrementing and adding the values from the given number till it reaches zero

Algorithm

Step 1:Number of elements stored in memory location is moved to accumulator
Step 2: Content of A is moved to R4
Step 3: Clear A
Step 4: Increment DPTR to get the first number
Step 5: Sum=0
Step 6: Carry=0
Step 7: Number is moved from DPTR to A
Step 8: Content of Register is moved to A
Step 9: Increment DPTR to get the next number
Step 10: Partial sum is moved to Register
Step 11: If carry jump to step13 else jump to step 12
Step 12: Jump to step 16
Step 13: Move R1 to A
Step 14: Add 01 to accumulator to increment the carry
Step 15: Carry is restored to R1
Step 16: Decrement and jump if R4 is not equal to zero to step 7 else moved to step 17.
Step 17: Sum is moved from Register to A
Step 18: Move sum from accumulator to memory location
Step 19: Increment DPTR in order to store the carry
Step 20: Carry is moved from R1 to A
Step 21: Store carry in to a memory location
Step 22: Halt the program

Result

Experiment 7:

MULTIPLICATION BY SHIFT AND ADD METHOD

AIM

Write a program to multiply two 8 bit numbers by shift and add method using 8051

APPARATUS REQUIRED

8051 Microcontroller kit, (0-5V) DC Power Supply

THEORY

Shift-and-add multiplication is similar to the multiplication performed by paper and pencil. This method adds the multiplicand X to itself Y times, where Y denotes the multiplier. To multiply two numbers by paper and pencil, the algorithm is to take the digits of the multiplier one at a time from right to left, multiplying the multiplicand by a single digit of the multiplier and placing the intermediate product in the appropriate positions to the left of the earlier results.

As an example, consider the multiplication of two unsigned 4-bit numbers, 8 (1000b) and 9 (1001b).Thus the multiplication can be performed by shifting and adding method. Shifting multiplier by one bit left and if the MSB is high, performs addition between product (intermediate) and multiplicand followed by shift. If MSB is low perform shifting only and the process continues for 2n times, where n is the number of bits in multiplier and multiplicand. The main advantage of this type process is its faster operation for large number of bit multiplication.

In general the multiplication require n-bit multiplicand by n-bit multiplier require 2n registers to hold numbers and product. And require 2n-bit adders and shifters.

An e.g. 4 bit multiplicand x 4-bit multiplier results 8-product and require 8-bit registers to hold data.

ALGORITHM

Step 1: Clear the product register
Step 2: Initialise counter register as 08
Step 3: load multiplicand to accumulator from 4200
Step 4: store multiplicand to R1
Step 5: load multiplier to accumulator from 4201 and store to R2
STEP 6: load product in R0 to accumulator
Step 7: rotate product left by one bit
Step 8: clear the LSB of product.
Step 9: store shifted product from A to Register
Step 10: load multiplier to accumulator and rotate multiplier through carry
Step 11: clear LSB of multiplier
Step 12: store the shifted multiplier to R2
Step 13: if no carry in shifting operation goto step 17
Step 14: load product to accumulator
Step 15: add product and multiplicand
Step 16: store result to product in Register

Step 17: decrement the counter R3 and if R3 not equal to zero goto step 6 else store the result
Step 18: store the result from Register to 4202
Step 19: halt the program

Result

Experiment**8:**

SQUARE, CUBE AND SQUARE ROOT

Aim:-

To find square, cube and square root of numbers using 8051.

ALGORITHM

Step1:Content from 4300(address of memory location whose square to be find out) is moved to accumulator and B register

Step 2: Multiply A and B

Step3:Store the lower eight bit of result to the memory location 4301

Step 4: move upper 8 bit from B to A

Step 5:Store upper 8 bit to the location 4302

Step 6:Hlt the program

Result

Experiment 9:

MATRIX ADDITION

AIM

To add two m x n matrices

Theory

By incrementing dptr and each time making change only in its most significant bit we can perform matrix (array) addition . Take values from 4300, 4400 and store added value in 4700 and increment to take values from 4301,4401 and store value in 4701 ,etc is the procedure followed. Each matrix to be added is placed as linear one dimensional arrays in 4300,4301,etc and other in 4400,4401,etc and values of added matrix is placed in similar fashion in 4700,4701,etc

ALGORITHM

Step 1:Move row value from memory location

Step2:Increment dptr to get column value

Step 3: Multiply row and column value to get total number of elements and store this value in Register

Step 4:Mov dpl 00

Step 5:Mov dph 43(dptr pointing element in first matrix)

Step 6:Move the element from address pointed by dptr

Step 7: Move this element to R1

Step 8: Move dph 44(dptr pointing to element of the second matrix)

Step 9:Move this element to accumulator

Step 10: Add two elements from two matrices

Step 11: Move dph 45(dptr pointing to element of the resultant matrix)

Step 12: Increment dptr

Step 13:decrement r0 and jump to step 5 if r0 is nonzero ;otherwise stop

Result

Experiment 10 .
LCM/HCF OF GIVEN NUMBERS

AIM

Write a program to find LCM/HCF of given numbers using 8051

THEORY

The least common multiple (also called the lowest common multiple or smallest common multiple) of two integers a and b, usually denoted by LCM (a, b), is the smallest positive integer that is a multiple of both a and b.

An example:

The LCM of 4 and 6:

Multiples of 4 are: 4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 56, 60, 64, 68, 72, 76…

And the multiples of 6 are: 6, 12, 18, 24, 30, 36, 42, 48, 54, 60, 66, 72...

Common multiples of 4 and 6 are simply the numbers that are in both lists: 12, 24, 36, 48, 60, 72...

So the least common multiple of 4 and 6 is the smallest one of those 12

The highest common factor (HCF), also known as the greatest common factor (GCF), or greatest common divisor (GCD), of two or more non-zero integers, is the largest positive integer that divides the numbers without a remainder.

An example: The number 54 can be expressed as a product of two other integers in several different ways:

54 X 1 = 27 X 2 = 18 X 3 = 9 X 6

Thus the divisors of 54 are:

1, 2, 3, 6, 9, 18, 27, 54

Similarly the divisors of 24 are:

1, 2, 3, 4, 6, 8, 12, 24

The numbers that these two lists share in common are the common divisors of 54 and 24:

1, 2, 3, 6

The greatest of these is 6.

That is the HCF of 54 and 24. One writes:

gcd(54, 24) = 6

ALGORITHM

Step 1:Move first number from memory location to register

Step2: Move second number from 4201 to r1

Step 3: Load number1 to accumulator

Step 4:Set R2 for LCM(first number)
Step5: Load number 2 to B
Step 6: Divide number1 by number2
Step 7:Move reminder to accumulator
Step 8:If acc=0,go to find HCF, else next step
Step 9:Move LCM to accumulator
Step 10:Acc=number1+LCM
Step 11: Store A to R2(as LCM)
Step 12:Go to next check
Step 13: Move number2 to accumulator
Step14: Set R3 for HCF and set number2 as HCF
Step 15: Load number2 to B
Step 16: Move number1 to A
Step 17: Divide number1/number2
Step 18: Move reminder to A
Step 19: If A=0,go to store result, else next step
Step 20:number2=reminder
Step 21: Move HCF to a
Step 22: Number1=HCF
Step 23:Go to next check
Step 24: Load LCM to A
Step 25: Point external memory location for storing LCM
Step 26: Store LCM
Step 27: Point external memory location for storing HCF
Step 28: Store HCF
Step 29 :Halt the program

Result

Experiment 11

Code conversion –Decimal/ASCII

AIM

To write programs to convert between hexadecimal, decimal and ASCII numbers.

THEORY

1. Acronym for the American Standard Code for Information Interchange. Pronounced ask-ee, ASCII is a code for representing English characters as numbers, with each letter assigned a number from 0 to127.
2. To get decimal value, 30H is subtracted from the ASCII code.

Decimal to ASCII

Step 1: Move the number from memory location to accumulator
Step 2: Add 30H to the content of accumulator
Step 3: store the value in to 420d
Step 4: Halt the program

Result

ASCII to Decimal

Step 1: Move the number from memory location to accumulator
Step 2: Subtract 30H from the content of accumulator
Step 3: store the value in to 420d
Step 4: Halt the program

Result

PART B

Experiment No. 1

Time Delay Generation and Relay Interface

AIM

To study time delay generation and relay interface using 8051

APPARATUS REQUIRED

8051 microcontroller kit, (0-5V) DC battery

THEORY

1. Assume the processor is clocked by a 12MHz crystal.
2. That means, the timer clock input will be 12MHz/12 = 1MHz
3. That means, the time taken for the timer to make one increment = 1/1MHz = 1uS
4. For a time delay of “X” uS the timer has to make “X” increments.
5. 2^16 = 65536 is the maximum number of counts possible for a 16 bit timer.
6. Let TH be the value that has to be loaded to TH register and TL be the value that has to be loaded to TL register.
7. Then, THTL = Hexadecimal equivalent of (65536-X) where (65536-X) is considered in decimal.

RESULT

Verified time delay generation and relay interface using 8051.

Experiment No : 2

Display (LED/Seven Segments/LCD) and Keyboard Interface

AIM

To write an assembly language program to display characters on a seven display interface.

APPARATUS REQUIRED

8051 microcontroller kit, (0-5V) DC battery

THEORY

- Enter a program.
- Initialize number of digits to Scan
- Select the digit position through the port address C0
- Display the characters through the output at address C8.
- Check whether all the digits are display.
- Repeat the Process.

Seven segment display

Digit	a	b	c	d	e	f	g
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1

- Form a 0 to 9 counter with a predetermined delay (around 1/2 second here).
- Convert the current count into digit drive pattern.
- Put the current digit drive pattern into a port for displaying.

SAMPLE INPUT AND OUTPUT:

Sl.No	Input (hex Values)	Output (Characters)

RESULT

Thus an assembly language program displaying characters on seven segment display has been executed.

Experiment No : 3

ADC INTERFACE

AIM

To write an assembly language program to display Characters on a seven display interface.

APPARATUS REQUIRED

8051 microcontroller kit, (0-5V) DC battery

THEORY

1. Make ALE low/high by moving the respective data from A register to DPTR.
2. Move the SOC(Start Of Conversion) data to DPTR from FFD0
3. Check for the End Of Conversion and read data from Buffer at address FFC0
4. **End the Program.**

RESULT

Thus an assembly language program is executed for analog to digital conversion.

Experiment No:4

DAC Interface with Waveform Generation

AIM: To write and execute 8051 programs to generate

SQUARE WAVE OF 50% DUTY CYCLE

Duty cycle =50%

$=T_{ON} / (T_{ON}+T_{OFF})$

Here $T_{ON} = T_{OFF}$

ALGORITHM:

Step1: move FF(analog voltage 10v) to accumulator

Step2: move accumulator to DAC input

Step3: move 00 (analog voltage 0v) to accumulator

Step4: move accumulator to DAC input and go to step1

SQUARE WAVE OF 40% DUTY CYCLE

Duty cycle =40% =0.4

$T_{ON} / (T_{ON}+T_{OFF}) = 0.4$

$T_{ON} = 0.4T_{ON} + 0.4T_{OFF}$

$0.6\ T_{ON} = 0.4T_{OFF}$

$T_{ON} / T_{OFF} = 2/3$

T_{ON} = 2 delay

T_{OFF} = 3 delay

ALGORITHM:

Step1: move FF(analog voltage 10v) to accumulator

Step2: move accumulator content to DAC input

Step3: move 00 (analog voltage 0v) to accumulator

Step4: move accumulator content to DAC input and go to step1

SAW TOOTH WAVEFORM

ALGORITHM:

Step1: move 00 (analog voltage 0v) to accumulator

Step2: move accumulator to DAC input

Step3: increment A

Step4: If A not equal to zero and go to step3 go to step1

TRIANGULAR WAVEFORM

ALGORITHM:

Step1: move 00 (analog voltage 0v) to accumulator

Step2: move accumulator to DAC input

Step3: increment A

Step4: If A not equal to FF (analog voltage 10V) go to step 2

Step5: decrement A

Step6: move accumulator to DAC input

Step7: : If A not equal to 0 (analog voltage 0V) go to step 5 else goto step 1

STAIRCASE WAVEFORM

Calculations:

No. of steps = 5

Step width = 255/5 = 51D= 33H

ALGORITHM:

Step1: move 00 (analog voltage 0v) to accumulator

Step2: move accumulator to DAC input

Step3: Add A and 33

Step4: Go to step1

Experiment No:5

Stepper Motor and DC Motor Interface

Aim: To write an assembly program to make the stepper motor and DC motor to run in forward and reverse direction.

APPARATUS REQUIRED

Stepper motor, 8051 microprocessor kit, (0-5V) power supply

THEORY

1. Fix the DPTR with the Latch Chip address FFC0
2. Move the values of register A one by one with some delay based on the 2-Phase switching Scheme and repeat the loop.
3. For Anti Clockwise direction repeat the step 3 by reversing the value sequence.
4. End the Program

360^0 clockwise direction

ADDRESS	LABEL	MNEMONICS
8000	AGAIN	MOV R6, #0C
8002	RPT1	MOV R0, #04
8004		MOV DPTR, #8200
8007		LCALL ROT
800A		DJNZ R6, RPT1
800C		SJMP AGAIN
800E	ROT	MOVX A, @DPTR
800F		MOV P1,A
8011		LCALL DELAY
8014		INC DPTR
8015		DJNZ R0, ROT
8017		RET

8018	DELAY	MOV R1, #0A
801A	D1	MOV R2, #FF
801C	D2	DJNZ R2, D2
801E		DJNZ R1, D1
8020		RET

360^0 COUNTER CLOCKWISE

ADDRESS	**LABEL**	**MNEMONICS**
8000	AGAIN	MOV R6, #0C
8002	RPT1	MOV R0, #04
8004		MOV DPTR, #8200
8007		LCALL ROT
800A		DJNZ R6, RPT1
800C		SJMP AGAIN
800E	ROT	MOVX A, @DPTR
800F		MOV P1,A
8011		LCALL DELAY
8014		INC DPTR
8015		DJNZ R0, ROT
8017		RET
8018	DELAY	MOV R1, #0A
801A	D1	MOV R2, #FF
801C	D2	DJNZ R2, D2
801E		DJNZ R1, D1
8020		RET

180 CLOCKWISE AND COUNTER CLOKWISE

ADDRESS	LABEL	MNEMONICS

8000	AGAIN	MOV R6, #06
8002	RPT1	MOV R0, #04
8004		MOV DPTR, #8200
8007		LCALL ROT
800A		DJNZ R6, RPT1
800C		MOV R6,#06
800E	RPT2	MOV R0,#04
8010		MOV DPTR,#8300
8003		LCALL ROT
8016		DJNZ R6, RPT2
8018		SJMP AGAIN
801A	ROT	MOVX A,@DPTR
801B		MOV P1,A
801D		LCALL DELAY
8020		INC DPTR
8021		RET
8023	DELAY	MOV R1, #50
8024	D1	MOV R2, #FF
8026	D2	DJNZ R2, D2
8028		DJNZ R1, D1
802B		RET

RESULT

Thus an assembly language program to control of stepper motor was executed successfully using 8051 Microcontroller kit.

Experiment No: 6

REALIZATION OF BOOLEAN EXPRESSION

AIM

Write an assembly language program to perform logical operations AND, OR, XOR on two eight bit numbers stored in internal RAM locations 21h, 22h

APPARATUS REQUIRED

8051 microcontroller kit, (0-5V) DC battery

Truthtable

C	B	A	S	C
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

RESULT

Verified Realization of Boolean expression through port

INSTRUCTION SET

ACALL target address
Function: Absolute Call
Flags: None
ACALL stands for "absolute call." It calls subroutines with a target address within 2K bytes from the current program counter (PC).
Eg. ACALL delay

ADD A, source byte
Function: ADD
Flags: OV, AC, CY
This adds the source byte to the accumulator (A), and places the result in A. Since register A is one byte in size, the source operands must also be one byte.
The ADD instruction is used for both signed and unsigned numbers.
Unsigned addition
In the addition of unsigned numbers, the status of CY, AC, and OV may change. The most important of these flags is CY. It becomes 1 when there is a carry from D7 bit
Example:

```
MOV A, #45H          ;A=45H
ADD A, #4FH                ;A= 94H (45H+4FH)
                           ;CY=0, AC=1
```

Addressing modes
The following addressing modes, are supported for the ADD instruction:

1. Immediate :ADD A,#data Example: ADD A,#25H
2. Register : ADD A, Rn Example: ADD A,R3
3. Direct: ADD A, direct Example: ADD A,30H
4. Register-indirect: ADDA,@Ri Examples: ADD A,@R0

Signed addition and negative numbers
In the addition of signed numbers, special attention should be given to the overflow flag (OV) since this indicates if there is an error in the result of the addition. There are two rules for setting OV in signed number operation. The overflow flag is set to 1:

- If there is a carry from D6 to D7 and no carry from D7 out.
- If there is a carry from D7 out and no carry from D6 to D7.
- Notice that if there is a carry both from D7 out and from D6 to D7, OV - 0.

Example:

```
MOV  A,# + 8         ;A= 0000 1000
MOV  Rl,#+4          ;R1=0000 0100
ADD  A,R1            ;A=0000 1100 OV=0,CY=0
```

Notice that D7 = 0 since the result is positive and OV = 0 since there is neither a carry from D6 to D7 nor any carry beyond D7. Since OV = 0, the result is correct [(+8) + (+4) = (+12)].

Example :

```
MOV  A, #+66         ;A=0010

MOV R1, #+4          ;R4 = 0100 0101
ADD A, R4            ;A=10 00 0111 = -121
                     ;(INCORRECT) CY=0, D7=l, OV=l
```

In the above example, the correct result is +135 [(+66) + (+69) = (+135)), but the result was -121. OV = 1 is an indication of this error. Notice that D7 = 1 since the result is negative; OV = 1 since there is a carry from D6 to D7 and CY = 0.

Example:

```
MOV A,#-126        ;      A = 1000 0010
MOV R7,#-127       ;      R7=1000 0001
ADD A,R7           ;      A=0000 0011 (+3, wrong)
                   ;      D7=0, OV = 1
```

CY = 1 since there is a carry from D7 out but no carry from D6 to D7.

From the above discussion we conclude that while CY is important in any addition, OV is extremely important in signed number addition since it is used to indicate whether or not the result is valid.

ADDC A, source byte

Function : Add with carry

Flags: OV, AC, CY

This will add the source byte to A, in addition to the CY flag (A = A + byte + CY). If CY = 1 prior to this instruction, CY is also added to A. If CY = 0 prior to the instruction, source is added to destination plus 0. This is used in multi byte additions. In the addition of 25F2H to 3189H, for example, we use the ADDC instruction as shown below.

Example:

```
CLR C              ; CY = 0
MOV A, #89H        ; A=89H
ADDC A, #0F2H      ; A = 89H+F2H+0=17BH, A = 7B, CY = 1
MOV R3,A           ; Save A
MOV A, #31H
ADDC A, #25H       ; A= 31H+25H+1=57H
```

Therefore the result is:

```
25F2H
+ 3199H
577BH
```

The addressing modes for ADDC are the same as for "ADD A, byte".

AJMP target address

Function : Absolute jump

Flag : None

AJMP stands for "absolute jump." It transfers program execution to the target address unconditionally. The target address for this instruction must be within 2K bytes of program memory.

ANL dest-byte, source-byte

Function : Logical AND for byte variables

Flags : None affected

Example :

```
MOV   A, #32H      ; A=32H            32     0011   0010
MOV   R4, #50H     ;R4=50H            50     0101   0000
```

ANL A, R4 ;(A=10H) 10 001 0000

For the ANL instruction there are a total of six addressing modes. In four of them, the accumulator must be the destination. They are as follows.

1. Immediate ANL A, # data Example ANL A, #25H
2. Register ANL A, Rn Example ANL A R3
3. Direct ANL A, direct Example ANLA, 30H;
4. Register – indirect : Example ANL A, @R0;
5. ANL direct/#data

Example: Assume that RAM location 32H has the value 67H. Find its content after execution of the following code.
ANL 32H,#44H
44H 0100 0100
67H 0110 0111
44H 0100 0101 Therefore, it has 44H.

ANL C, source-bit
Function:Logical AND for bit variable
Flag: CY

In this instruction the carry flag bit is ANDed with a source bit and the result is placed in carry. Therefore, if source bit = 0, CY is cleared; otherwise, the CY flag remains unchanged.

CJNE dest – byte, source – byte, target

Function : Compare and jump if not equal
Flag : CY
The magnitudes of the source byte and destination byte are compared. If they are not equal, it jumps to the target address.
Example : Keep monitoring P1 indefinitely for the value of 99H. Get out only when P1 has the value 99H.

```
MOV   Pl, OFFH        ;make Pl an input port
Back  MOV A, Pl       ; read Pl
CJNE  A, #99, Back    ;keep monitoring
```

Notice that CJNE jumps only for the not-equal value. To find out if it is greater or less after the comparison, we must check the CY flag. Notice also that the CJNE instruction affects the CY flag only, and after the jump to the target address the carry flag indicates which value is greater, as shown here.
In the following example, Pl is read and compared with value 65. Then:

Dest<Source CY = 1
Dest>Source CY=0

1. If Pl is equal to 65, the accumulator keeps the result.
2. If Pl has a value less than 65, R2 has the result, and finally
3. If Pl has a value greater than 65, it is kept by R3.

At the end of the program, A will contain the equal value, or R2 the smaller value, or R3 the greater value.
Example :

```
MOV     A, Pl           ; Read Pl
CJNE    A, #65, NEXT    ; Is it 65
        SJMP EXIT       ;yes,A keeps it,EXIT
NEXT :  JNC OVER        ; NO
MOV     R2, A           ;Save the smaller in R2
SJMP    EXIT            ;And EXIT
OVER:   MOV  R3, A      ;Save the larger in R3
EXIT :
```

This instruction supportsn four addressing modes. In two of them, A is the destination.

1. Immediate CJNE A, #data target

Example CJNE A, #96, NEXT ; JUMP IF A IS NOT 96

2. Direct CJNE A, direct, target ; Jump If A Not

; with the value held by RAM LOC. 40H

Notice the absence of the "#" sign in the above instruction. This indicates RAM location,,40H. Notice in this mode that we can test the value at an input port. This is a widely used application of this instruction. See the following:

```
MOV   PI, OFF       ; PI is an input port
MOV   A, #10H       ;A = 10H
HERE: CJNE  A, PI,HERE   ;WAIT HERE TIL PI = 10H
```

In the third addressing mode, any register, RO - R7, can be the destination.

3. Register: CJNE Rn, #data, target

Example: CJNE R5,#70,NEXT ;jump if R5 is not 70

In the fourth addressing mode, any RAM location can be the destination. The RAM location is held by register ROorRl.

4. Register-indirect: CJNE @ Ri, #data, target

Example: CJNE @R1, #80, NEXT ; jump if RAM
;location whose address is held by Rl
; is not equal to 80

Notice that the target address can be no more than 128 bytes backward or 127 bytes forward, since it is a 2-byte instruction.

CLR A

Function: Clear accumulator

Flag: None are affected

This instruction clears register A. All bits of the accumulator are set to 0.

CLR bit

Function : Clear bit

This instruction clears a single bit. The bit can be the carry flag, or any bit – addressable location in the 8051. Here are some examples of its format:

```
CLR  C          ;CY=0
CLR  P2.4       ;CLEAR P2.4  (P2.4=0)
CLR  Pl.7       ;CLEAR P1.7 (P1. 7=0)
LR   ACC.7      ; CLEAR D7 of ACCUMULATOR (ACC. 7=0)
```

CPL A

Function: Complement accumulator
Flags: None are affected

This complements the contents of register *A,* the accumulator. The result is the l's complement of the accumulator. That is: Os become 1s and 1s become Os.

Example:

```
MOV  A, #55H     ; A=01010101
AGAIN :CPL  A    ; compliment reg. A
MOV  P1, A       ; toggle bits of Pl
SJMP  AGAIN      ;Continuously
```

CPL bit

Function : Complement bit

This instruction complements a single bit. The bit can be any bit-addressable location in the 8051.

Example:

```
SETB  P1. O      ;set PI.0 high
AGAIN: CPL  PI. 0  ; complement port. bit
SJMP  AGAIN      ; continuously
```

D A A

Function: Decimal-adjust accumulator after addition
Flags: CY
This instruction is used after addition of BCD numbers to convert the result back to BCD. The data is adjusted in the following two possible cases.

1. It adds 6 to the lower 4 bits of A if it is greater than 9 or if AC = 1.
2. It also adds 6 to the upper 4 bits of A if it is greater than 9 or if CY = 1.

Example

```
MOV  A, # 47H     ; A=0100 0111
ADD  A, #38H      ;A=47H+38H=7FH, invalid BCD
DA   A      ;A=1000 0101=85H, valid BCD
```

47 H
+38H

7FH (invalid BCD)
+6H (after DA A)
85H (valid BCD)
In the above example, since the lower nibble was greater than 9, DAA added 6 to A. If the lower nibble is less than 9 but AC = 1, it also adds 6 to the lower nibble. See the following example.
Example:

```
MOV  A,#29H      ;A=0010      1001
ADD  A,#18H      ;A= 0100     0001   INCORRECT
DA   A           ;A= 0100     0111   = 47H VALID BCD
```

29H
+ 18H
41H (incorrect result in BCD)
+6H
47H correct result in BCD

The same thing can happen for the upper nibble. See the following example.

Example:

```
MOV  A,#52H              ;A=0101 0010
ADD  A, #91H     ;A=1110 0011 Invalid BCD   1001   0001
                 ;A=0100      0011   AND CY = 1
DA   A
```

52H
+ 91H
E3H (invalid BCD)
+ 6 (after DA A, adding toupper nibble)
143H valid BCD

Similarly, if the upper rubble is less than 9 and CY = 1, it must be corrected. See the following example.

Example:

```
MOV  A, #54H     ;A= 0101 0100
ADD  A, #87H     ;A=1101 1011 INVALID BCD
DA   A           ;A=0100 0001, CY=1 (BCD 141)
```

DEC byte
Function : Decrement
Flags : None
This instruction subtracts 1 from the byte operand. Note that CY (carry/borrow) is unchanged even if a value 00 is decremented and becomes FF. This instruction supports four addressing modes.

1. Accumulator DEC A Example : DEC A
2. Register DEC Rn Example : DEC R1 or DEC R3
3. Direct : DEC direct Example : DEC 40H
4. Register-indirect: DEC@Ri ;where i=0 or 1 only

;Example :DEC @R0

DIV AB

Function : Divide

Flags CY and OV

This instruction diveds a byte in accumulator by the byte in register. B. It is assumed that both registers A and B contain an unsigned byte.After the division, the quotient will be in register A and the remainder in register B. If you divide by zero (that is, set register B = 0 before the execution of "DIV AB" the values in register A and B are undefined and the OV flag is set to high to indicate in invalid result. Notice that CY is always 0 in this instruction.

Example:

```
MOV A,#35
MOV B, #10
DIV AB          ;A=3 and B=5
```

Notice in this instruction that the carry and OV flags are both cleared, unless we divide A by 0, in which case the result is invalid and OV = 1 to indicate the invalid condition.

DJNZ byte, target

Function: Decrement and jump if not zero

Flags: None

In this instruction a byte is decremented, and if the result is not zero it will, jump jo the target address.

Example: Count from 1 to 20 and send the count to PI.

```
        CLR A               ;A=0
        MOV R2,#20          ;R2=20 counter
BACK: INC   A
        MOV PI, A
DJNZ R2, BACK               ; repeat if R2 not = zero
```

The following two formats are supported by this instruction

1. Register : DJNZ Rn, target (where n = 0 to 7) Example DJNZ R3, Here
2. Direct : DJNZ direct, target

Notice that the target address can be no more than 128 bytes backward or 127 bytes forward, since it is a 2-byte instruction.

INC byte

Function: Increment

Flags: None

This instruction adds 1 to the register or memory location specified by the operand. Note that CY is not affected even if value FF is incremented to 00. This instruction supports four addressing modes.

Accumulator: INC A Example: INC A

Register: INC Rn Example: INC R1 or INC R5

INC Direct: Example: INC 30H

Register-indirect: INC @Ri(i=0or1) Example: INC @R0 ;

INC DPTR
Function: Increment data pointer
Flags: None
This instruction increments the 16-bit register DPTR (data pointer) by 1. Notice that DPTR is the only 16-bit register that can be incremented. Also notice that there is no decrement version of this instruction.
Example:

```
MOV   DPTR,#16FFH          ;DPTR=16FFH
INC     DPTR  ; now DPTR=1700H
```

JB bit,target also: JNB bit,target
Function: Jump if bit set Jump if bit not set
Flags: None

These instructions are used to monitor a given bit and jump to a target address if a given bit is high or low. In the case of JB, if the bit is high it will jump, while for JNB if the bit is low it will jump. The given bit can be any of the bit- addressable bits of RAM, ports, or registers of the 8051.

Example: Monitor bit P1.5 continously. When it becomes low, send 55H to P2.

```
SETB PI.5       ;make PI.5 an input bit
HERE: JB        PI. 5, HERE    ;stay here as    long     as PI. 5=1
MOV   P2,#55H          ; since PI.5=0  send    55H to P2

JNB     ACC. 0, NEXT          ;jump  if DO  is        0 (even)
INC     A                     ;D0-1,  make   it       even
NEXT:
```

JBC bit,target
Function: Jump if bit is set and clear bit
Flags: None

If the desired bit is high it will jump to the target address; at the same time the bit is cleared to zero.

Example: The following instruction will jump to label NEXT if D7 of register A is high; at the same time D7 is cleared to zero.

```
JBC     ACC.7,NEXT
MOV   PI,A
NEXT:
```

Notice that the target address can be no more than 128 bytes backward or 127 bytes forward since it is a 2-byte instruction.
JC target
Function: Jump if CY = 1.
Flags: None
This instruction examines the CY flag; if it is high, it will jump to the target address.

JMP @A+DPTR
Function: Jump indirect

Flags: None

The JMP instruction is an unconditional jump to a target address.

The target address is provided by the total sum of register A and the DPTR register. Since this is not *a* widely used instruction we will bypass further discussion of it.

JNB bit,target
See JB and JNB.
JNC target
Function: Jump if no carry (CY = 0)
Flags: None
JNZ target
Function: Jump if accumulator is not zero
Flags: None
This instruction jumps if register A has a value other than zero.

JZ target
Function: Jump if A =zero
Flags: None

This instruction examines the contents of the accumulator and jumps if it has value 0.

Notice that this is a 2-byte instruction; therefore, the target address cannot be more than-128 to +127 bytes away, from the program counter. See J condition for further discussion on this.

J condition target
Function: Conditional jump

In this type of jump, control is transferred to a target address if certain conditions are met. The target address cannot be more than -128 to +127 bytes away from the current PC (program counter).

JC	Jump carry	jump if CY = 1
JNC	Jump no carry	jump if CY = 0
JZ	Jump zero	jump if register A = 0
JNZ	Jump no zero	jump if register A.is not 0
JNB bit	Jump no bit	jump if bit = 0
JB bit	Jump bit	jump if bit = 3
JBC bit	Jump bit clear bit	jump if bit = 1 and clear bit
DJNZ Rn,...	Decrement and jump if not zero	
CJNE A,#val,...	Compare A with value and jump if not equal	

LCALL 16-bit addr
Function: Transfers control to a subroutine
Flags-. None

There are two types of CALLs: ACALL and LCALL. In ACALL, the target address is within 2K bytes of the current PC (program counter). To reach the target address in the 64K bytes maximum ROM space of the 8051, we must use LCALL. If calling a subroutine, the PC

register (which has the address of the instruction after the ACALL) is pushed onto the stack, and the stack pointer (SP) is incremented by 2. Then the program counter is loaded with the new address and control is transferred to the subroutine. At the end of the procedure, when RET is executed, PC is popped off the stack, which returns control to the instruction after the CALL.

Notice that LCALL is a 3-byte instruction, in which one byte is the opcode, and the other two bytes are the 16-bit address of the target subroutine. ACALL is a 2-byte instruction, in which 5 bits are used for the opcode and the remaining 11 bits are used for the target subroutine address. An 11-bit address limits the range to 2K bytes.

LJMP 16-bit addr

Function: Transfers control unconditionally to a new address.

In the 8051 there are two unconditional jumps: LJMP (long jump) and SJMP (short jump). Each is described next.

1. LJMP (long jump): This is a 3-byte instruction. The first byte is the opcode and the next two bytes are the target address. As a result, LJMP is used to jump to any address location within the 64K-byte code space of the 8051. Notice that the difference between LIMP and LCALL is that the CALL instruction will return and continue execution with the instruction following the CALL, whereas JMP will not return.
2. SJMP (short jump): This is a 2-byte instruction. The first byte is the opcode and the second byte is the signed number displacement, which is added to the PC (program counter) of the instruction following the SJMP to get the target address. Therefore, in this jump the target address must be within -128 to +127 bytes of the PC (program counter) of the instruction after the SJMP since-a single byte of address can take values of +127 to —128. This address is often referred to as *relative address* since the target address is -128 to +127 bytes relative to the program counter (PC). In this Appendix, we have used the term target address in place of relative address only for the sake of simpilicity.

MOV dest-byte/source-byte

Function: Move byte variable Flags: None

This copies a byte from the source location to the destination. There are fifteen possible combinations for this instruction. They are as follows:

(a) Register A as the destination. This can have the following formats.

MOV A,#data Example: MOVA,#25H ;(A=25H)

MOV A,Rn Example: MOV A,R3

MOV A,direct Example: MOV A, 3 OH ;A= data in 3OH

MOV A,@Ri (i=0 or 1)Examples: MOV A, @RO

Notice that "MOV A, A" is invalid.

(b) Register A is the source. The destination can take the following forms

5. MOV Rn. A
6. MOV direct, A
7. MOV @Ri.A

(c) Rn is the destination

8. MOV Rn, # immendiate
9. MOV Rn, A
10. MOV Rn, direct

(d) The destination is a direct address
11. MOV direct, # data
12. MOV direct, @Ri
13. MOV direct, A
14. MOV direct , Rn
15. MOV direct, direct

(e) Destination is an indirect address held by R0 or R1
16. MOV @ Ri #data
17. MOV @ Ri, A
18. MOV @Ri direct

MOV dest – bit, source – bit
Function : Move bit data
This MOV instruction copies the source bit to the destination bit. In this instruction one of the operands must be the CY flag. Look at the following examples.

```
MOV P1. 2, C        ;Copy carry bit to port bit Pl.2
MOV C, P2.5         ;copy port bit P2.5 to carry bit
```

MOV DPTR, #16 – bit value
Function : Load data pointer
Flags : None
This instruction loads the 16-bit DPTR (data pointer) register with a 16-bit immediate value
Examples
MOV DPTR, # 456FH ;DPTR-456FH
MOV C A ,@A+DPTR
Function : Move code byte
Flags : None

This instruction moves a byte of data located in program (code) ROM into register A. This allows us to put strings of data, such as look-up table elements, in the code space and read them into the CPU. The address of the desired byte in the code space (on-chip ROM) is formed by adding the original value of the accumulator to the 16-bit DPTR register.

MOVC A,@A+PC
Function: Move code byte
Flags: None

This instruction moves a byte of data located in the program (code) area to A. The address of the desired byte of data is formed by adding the program counter (PC) register to the original value of the accumulator. Contrast this instruction with "MOVC A, @A+DPTR". Here the PC is used instead of DPTR to generate the data address.

MOVX dest-byte, source-byte
Function: Move external
Flags: None

This instruction transfers data between external memory and register A.
Example MOVX A,@DPTR

This moves into the accumulator a byte from external memory whose address is pointed to by DPTR. In other words, this brings data into the CPU (register A) from the off-chip memory of the 8051.

MOVX @DPTR,A
This moves the contents of the accumulator to the external memory location whose address is held by DPTR. In other words, this takes data from inside the CPU (register A) to memory outside the 8051.

(a) The 8-bit address of external memory is held by RO or Rl.

MOVX A, @Ri ;wherei = 0 or 1

This moves to the accumulator a byte from external memory whose 8-bit address is pointed to by RO (or Rl in MOVX A,@R1).

MOVX @Ri,A

This moves a byte from register A to an external memory location whose 8-bit address is held by R0(or R1 in MOVX @R1.A)

The16-bit address version of this instruction is widely used to access external memory while the 8-bit version is used to access external 1/O ports.

MUL AB

Function: Multiply AxB
Flags: OV, CY

This multiplies an unsigned byte in A by an unsigned byte in register B. The result is placed in A and B where A has the lower byte and B has the higher byte.
Example:
MOV A, #5
MOV B,#7
MUL AB ;A=35=23H, B=00

NOP
Function: No operation
Flags: None

This performs no operation and execution continues with the next instruction. It is sometimes used for timing delays to waste clock cycles. This instruction only updates the PC (program counter) to point to the next instruction following NOP.
ORL dest-byte,source-byte
Function: Logical OR for byte variable
Flags: None
This performs a logical OR on the byte operands,
bit by bit, and stores the result in the destination.
For the ORL instruction there are a total of six addressing modes. In four of them the accumulator must be the destination. They are as follows:

1. Immediate: ORL A,#data Example: ORL A,#25H
2. Register: ORL A,Rx Example: ORL A, R3
3. Direct: ORL A,direct Example: ORL A, 30H;
4. Register-Indirect: ORL A,@Rn Example: ORL A,@R0

In the next two addressing modes the destination is a direct address (a RAM location or one of the SFR registers), while the source is either A or immediate data as shown below:

5. ORL direct,“data”

Example: Assuming that RAM location 32H has the value 67H, find the content of A after the following:

```
ORL 32H,#44H      ;OR 44H with contents of RAM 1oc. 32H
MOV A, 32H        ;move content of RAM loc. 32H to A
```

6. ORL direct,A

ORL C, source-bit

Function: Logical OR for bit variables
Flags: CY
In this instruction the carry flag bit is ORed with a source bit and the result is placed in the carry flag. Therefore, if the source bit is 1, CY is set; otherwise, the CY flag remains unchanged.

POP direct

Function: Pop from the stack
Flags: None

This copies the byte pointed to by SP (stack pointer) to the location whose direct address is indicated, and decrements SP by 1. Notice that this instruction supports only direct addressing mode. Therefore, instructions such as "POP A" or "POP R3 " are illegal. Instead we must write "POP OEOH" where EOH is the RAM address belonging to register A and "POP 03 " where 03 is the RAM address of R3 of bank 0.

PUSH d irect

Function:Push onto the stack
Flags: None
This copies the indicated byte onto the stack and increments SP'by 1. Notice that this instruction supports only direct addressing mode. Therefore, instructions such as "PUSH A" or "PUSH R3" are illegal. Instead, we must write "PUSH OEOH" where EOH is the RAM address belonging to register A and "PUSH 03 " where 03 is the RAM address. of R3 of bank 0.

RET

Function: Return from subroutine
Flags: None
This instruction is used to return from a subroutine previously entered by instructions LCALL or ACALL. The top two bytes of the stack are popped into the program counter (PC) and program execution continues at this new address. After popping the top two bytes of the stack into the program counter, the stack pointer (SP) is decremented by 2.

RFTI

Function: Return from interrupt
Flags: None

This is used at the end of an interrupt service routine (interrupt handler). The top two bytes of the stack are popped into the program counter and program execution continues at this new address. After popping the top two bytes of the stack into the program counter (PC), the stack pointer (SP) is decremented by 2.

RL A
Function:Rotate left the accumulator
Flags: None

This rotates the bits of A left. The bits rotated out of A are rotated back into A at the opposite end.
Example:

```
MOV A, #69H; A=01101001
RL      A               ; Now A=11010010
RL A                    ; Now A = 10100101
```

RLC A
Function: Rotate A left through carry
Flags: CY
This rotates the bits of the accumulator left. The bits rotated out of register A are rotated into CY, and the CY bit is rotated into the opposite end of the accumulator.
Example :

```
CLR   C       ;CY=0
MOV   A, #99H       ;A-10011001
RL A                ; Now A=00110010 and CY=1
RL      A           ; Now A=10100101
```

RLC A
Function: Rotate A right
Flags: None
This rotates the bits of register A right. The bits rotated out of A are rotated back into A at the opposite end.

Example:

```
Now A         #66H          ; A=01100110
RR      A                   ;Now A=00110011
RR      A                   ;Now A=10011001
```

RRC A
Function:Rotate A right through carry
Flags: CY

This rotates the bits of the accumulator right. The bits rotated out of register A are rotated into CY and the CY bit is rotated into the opposite end of the accumulator.

SETB bit
Function: Set bit

This sets high the indicated bit. The bit can be the carry or any directly addressable bit of a port, register, or RAM location.
Examples :

```
SETB Pl.3          ;      pl. 3=1
SETB P2.6          ;P2.6=1
SETB ACC.6         ;ACC, 6=1
SETB 05            ;Set high D5 of RAM loc. 20H
SETB C             ; Set carry Flag CY = 1
```

SJMP
See LJMP & SJMP.

SUBB A, source byte
Function: Subtract with borrow
Flags: OV, AC, CY

This subtracts the source byte and the carry flag from the accumulator and puts the result in the accumulator; The steps for subtraction performed by the internal hardware of the CPU are as follows:

1. Take the *2's* complement of the source byte.
2. Add this to register A,
3. Invert the carry.

This instruction sets the carry flag according to the following:

Dest> source	0	the result is positive
Dest=source	0	the result is 0
Dest<source	1	the result is negative in 2's complement

Notice that there is no SUB instruction in the 8051. Therefore, we perform the SUB instruction by making CY = 0 and then using SUBB: *A = (A*- byte - CY).

Addressing Modes
The following four addressing modes are supported for the SUBB

1. Immediate SUBB A, # data Example : SUBB A, #25H ;A=A-25H-CY
2. Register SUBB A, Rn Example : SUBB A, R3 ;A=A-R3-CY
3. Direct : SUBB A, direct Example : SUBB A, 30H ;A=data at (30H)-CY
4. Register-indirect : SUBB A, @ Rn Example : SUBB A, @R0 A=data at (R0)-CY

SWAP A
Function : Swap nibbles within the accumulator
Flags : None
The SWAP instruction interchange the lower ribble (D0-D3) with the upper ribble (D4-D7) inside register A.
Example

```
MOV A, #59H;A=59H (0101   1001  in binary)
SWAP A             ;A=95H (1001  0101  in binary)
```

XCH A, Byte
Function : Exchange A with a byte variable

Flags : None
This instruction swaps the contents of register A and the source byte. The source byte can be any register or RAM location.
Example

```
MOV   A, #65H          ;A=65H
MOV   R2, #97H         ;R2=97H
XCH   A, R2            ;Now A=97H and R2=65H
```

For the "XCHA, byte' instruction there are a total of three addressing modes. They are as follows

1. Register XCH A, Rn Example XCH A, R3
2. Direct XCH A, direct; Example XCH A, 40H Register indirect: XCH A, @ Rn;Examples XCH A @R0 ;

XCHD A, @R1
Function : Exchange digits
Flags: None
The XCHD instruction exchanges only the lower ribble of A with the lower ribble of the Ram location pointed to by Ri while leaving the upper ribbles in both places intact.
Example : Assuming RAM location 40H has the value 97H find its content after the following instructions.

```
;40H = (97H)
MOV A, #12H        ;A = 12H (0001  0010   binary)
MOV   R1 #40H          ;R1=40H, load pointer
XCHD A @R1             ;exchange the lower nibble of
                       ; A and RAM location 40H
```

After execution of the XCHD instruction, we have A = 17H and RAM location 40H has 92H.

XRL dest-byte,source-byte
Function:Logical exclusive-OR for byte variables
Flags: None

This performs a logical exclusive-OR on the operands, bit by bit, storing the result in the destination.
Example:

```
MOV A,#39H         ; A = 39H
XRL A,#09H             ; A = 39H ORed with 09

39H    0011   1001
09H    0000 1001
30     0011 0000
```

For the XRL instruction there are total of six addressing modes. In four of them the accumulator must be the destination. They are as follows:

1. Immediate: XRL A,"data:" Example: XRL A,#25H
2. Register: XRL A,Rn Example: XRL A,R3
3. Direct: XRL A,direct ;XRL A with data in RAM location 3OH ;
4. Register-indirect XRL A,@Rn ;Example: XRL A,@R0
   ;XRL A with data pointed to by RO

In the next two addressing modes the destination is a direct address (a RAM location or one of the SFR registers) while the source is either A or immediate data as shown below:

5. XRL direct,#data
Example: Assume that RAM location 32H has the value 67H.
Find the content of A after execution of the following code.
XRL 32H,#44H ;move content of RAM loc . 3 2H • to A
MOV *A*, 32H

44H 0100 0100
67H 0110 0111
23H 0010 0011 Therefore A will have 23H.

6.XRL direct, A