G. NARAYANAMMA INSTITUTE OF TECHNOLOGY AND SCIENCE

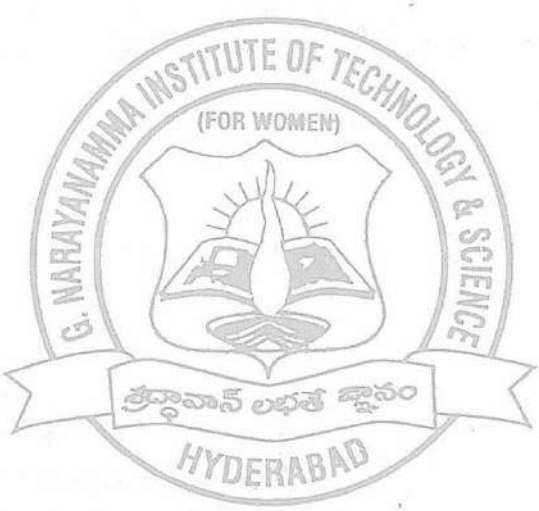
[AUTONOMOUS]

Accredited by NBA & NAAC, Affiliated to

JNTUH, Shaikpet, Hyderabad-104

YEAR: III-B.Tech I & II-SEM A.Y: 2023-2024

MICROPROCESSORS & MICROCONTROLLERS LAB MANUAL



Name of Student: ……

Roll NO :……

Branch: …….. Section: ……..

Year: …… Semester……..

Department of Electrical & Electronics Engineering

NBA & NAAC Accredited

Recognized Research Centre by JNTUH

PREFACE

Electrical engineering is a fundamental discipline that underpins many aspects of our technologically advanced world. It encompasses the study and application of electrical principles, circuits, and devices, which are essential in various industries and everyday life. The Basic Electrical Engineering Laboratory provides students with a hands-on opportunity to explore and experiment with the fundamental concepts and components that form the foundation of electrical engineering.

This laboratory course is designed to introduce students to the fundamental principles of electrical engineering and to equip them with practical skills that will be invaluable throughout their academic and professional journeys. By conducting experiments, analyzing data, and troubleshooting circuits, students will gain a deeper understanding of basic electrical concepts and build a strong foundation for more advanced coursework in the field.

By actively engaging in these laboratory exercises and following the outlined procedures, you will not only strengthen your understanding of basic electrical engineering but also develop the skills and knowledge necessary to excel in more advanced electrical engineering courses and real-world applications. Electrical engineering is a field with boundless opportunities, and this laboratory experience is the first step in your exciting journey.

HOD-EEE

Course Objectives:

1. To infer the basics of the microprocessor and its assembly language.
2. To extend the basics of assembly language to the microcontroller.
3. To provide foundation on interfacing the external devices to the micro controller.
4. To develop solutions for the real time applications.

Course Outcomes:

At the end of this course, students will be able to:

1. Illustrate the assembly language programming.
2. Design circuits for various applications using microcontroller.
3. Apply the concepts of microcontroller on real- time applications.
4. Evaluate the results of 8086 and 8051 programs.
5. Use standard test and measurement equipment to evaluate analog/digital interfaces.
6. Analyze abstract problems and apply a combination of hardware and software to address the problem.

CO-PO Mapping Matrix:

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
CO 1	2													
CO 2		2	3											
CO 3			3	1					2					
CO 4			2											
CO 5	1	2												
CO					2									

Program Outcomes:

PO1 Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.

PO2 Problem analysis: Identify, formulate, review research literature and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.

PO3 Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety and the cultural, societal, and environmental considerations.

PO4 Conduct investigations of complex problems: Use research – based knowledge and research methods including design of experiments, analysis and interpretation of data and synthesis of the information to provide valid conclusions.

PO5 Modern tool usage: Create, select and apply appropriate techniques, resources and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6 The engineer and society: Apply reasoning informed by the contextual knowledge to asses societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7 Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts and demonstrate the knowledge of, and need for sustainable development.

PO8 Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9 Individual and teamwork: Function effectively as an individual and as a member or leader in diverse teams and in multidisciplinary settings.

PO10 Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and

write effective reports and design documentation, make effective presentations and give and receive clear instructions.

PO11 Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team ,to manage projects and in multidisciplinary environments.

PO12 Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Safety Instructions to Students

Do's:	Don'ts:
Follow Lab Guidelines: Adhere to the lab guidelines and instructions provided by the instructor. **Check Equipment:** Before starting any experiment, check the condition of equipment and ensure that it is functioning properly. **Document Procedures:** Document the procedures and results systematically in a lab notebook. **Backup Work:** Regularly backup your work and code to prevent data loss. **Ask Questions:** Don't hesitate to ask questions if you're unsure about any aspect of the experiment or equipment usage. **Collaborate Responsibly:** Collaborate with lab partners, but ensure that each student understands and contributes to the work. **Keep the Workspace Tidy:** Keep the work area clean and organized. **Report Issues:** Report any malfunctioning equipment or safety concerns to the instructor. **Save Data:** Save important data and code regularly during experiments. **Respect Lab Time:** Make the most of the lab time allocated for experiments. Be focused and productive. **Follow Ethical Guidelines:** Adhere to ethical guidelines and academic honesty.	**Don't Rush:** Avoid rushing through experiments. Take the time to understand each step. **No Food or Drinks:** Do not bring food or drinks into the lab to prevent contamination and damage to equipment. **No Unauthorized Modifications:** Do not modify or tamper with equipment without permission. **Avoid Distractions:** Minimize distractions, such as unnecessary conversations or use of personal electronic devices. **Don't Skip Safety Procedures:** Follow safety procedures, and do not neglect the use of safety equipment. **Don't Panic:** If there's an issue, don't panic. Inform the instructor and seek assistance. **No Unauthorized Access:** Do not access areas or equipment that you are not authorized to use. **Don't Leave Equipment Unattended:** Do not leave equipment unattended, especially when it is powered on. **Avoid Overloading Circuits:** Do not overload circuits or violate specified operating conditions.

LIST OF EXPERIMENTS

Programming using 8086:

1. Arithmetic operations (Addition, Subtraction, Multiplication and Division).
2. Sorting of an Array.
3. Searching for characters in a string.
4. Program for string manipulations for 8086.

Programming using 8051:

5. Arithmetic operations (Addition, Subtraction, Multiplication and Division).
6. Write a program to interface LEDs to 8051.
7. Programming Timer/Counter using 8051.
8. Program to interface a Keyboard using 8051.
9. Write a program to verify Interrupt handling in 8051.
10. Program to implement UART operation to 8051.

Additional Experiments:

11. Interfacing LCD to 8051.
12. Write a program to interface stepper/DC motor with 8051.

GNITS- EEE DEPARTMENT

MICROPROCESSORS & MICROCONTROLLERS LAB

III B.Tech, II Semester

S.No	Date	Name of the Experiment	Page No	Grade	Signature
1.					
2.					
3.					
4.					
5.					
6.					
7.					
8.					
9.					
10.					
Additional Experiment					

1. a -8 BIT ADDITION

Aim: To perform 8 bit addition of two hexadecimal numbers using input memory offset address 20f0h and the result stored at output memory offset address 20f2h.

Apparatus: TASM, PC

Algorithm:

1. Start
2. Initialize the address of code segment register to data segment register to use tiny memory model.
3. Load first number to lower byte of accumulator register from given memory location.
4. Load the second number to lower byte of base register from given memory location.
5. Perform addition on first and second operands and store the result in destination operand.
6. Store the accumulator content to given memory location.
7. Set break point.
8. Exit from DOS prompt.
9. Stop

Program:

```
.model tiny
.stack 32h
.code
org 2000h
start:      mov ax,cs
            mov ds,ax
            mov ax,00h
            mov al,num1
            mov bl,num2
            add al,bl
            mov result,al
            int 3h
            mov ah,4ch
            int 21h
org 20f0h
            num1 db 03h
            num2 db 08h
            result db 00
end start
```

Result:

Without carry:

Input	Data	output	data
num1(20f0h)	03h	Result(20f2h)	0Bh
num2(20f1h)	08h	carry	0

With carry:

Input	Data	output	data
num1(20f0h)	0aah	Result(20f2h)	0a9h
num2(20f1h)	0ffh	carry	1

8 bit addition of two hexadecimal numbers has been performed.

Exercise:

1. What is the significance of model tiny?
2. How many model assignments are the name them?
3. What is a directive?
4. What is a pseudo operation?
5. ORG 2000H implies what?
6. At register is used why not AX?
7. What is the purpose of INT3 in the program?
8. What is the purpose of MOV AH, 4CH/ INT 21H in the program?

1.b - 16 BIT ADDITION

Aim: To perform 16-bit addition of two hexadecimal numbers without and with carry using input memory offset address is 20f0h and the result is stored at output memory offset address is 20f4h.

Apparatus: TASM, PC

Algorithm:

1. Start
2. Initialize the address of code segment register to data segment register to use tiny memory model.
3. Load the first number to accumulator register from given memory location.
4. Load the second number to base register from given memory location.
5. Perform addition on first and second operands and store the result in destination operand.
6. Store the accumulator content to given memory location.
7. Set break point.
8. Exit from DOS prompt.
9. Stop

Program:

```
.model tiny
.stack 32h
.code
org 2000h
start:      mov ax,cs
            mov ds,ax
            mov ax,00
            mov ax,num1
            mov bx,num2
            add ax,bx
            mov result,ax
            int 3
            mov ah,4ch
            int 21h
org 20f0h
            num1 dw 04526h
            num2 dw 01352h
            result dw 00
end start
```

Result:

Without carry:

Input	Data	output	data
num1(20f0h)	04526h	Result(20f4h)	05878h
num2(20f2h)	01352h	carry	0

With carry:

Input	Data	output	data
num1(20f0h)	0bafeh	Result(20f4h)	97ech
num2(20f2h)	0dceeh	carry	1

16 bit addition of two hexadecimal numbers has been performed.

Exercise:

1. What is significance of Stack 32h?
2. Can ORG have other numbers instead of 2000h?
3. What is the purpose of MOV AX, CS? MOV DS, AX?
4. Why AX register is used and not AL?
5. What is the purpose of DW?
6. What happens if the result is greater than 16bit when result is declared an DW?

1.c - 8 BIT SUBTRACTION

Aim: To perform 8 bit subtraction of two hexadecimal numbers without and with borrow using input memory offset address 20f0h and the result stored at output memory offset address 20f2h.

Apparatus: TASM, PC

Algorithm:

1. Start
2. Initialize the address of code segment register to data segment register to use tiny memory model.
3. Load first the number to lower byte of accumulator register from given memory location.
4. Load the second number to lower byte of base register from given memory location.
5. Perform subtraction on first and second operands and store the result in destination operand.
6. Store the accumulator content to given memory location.
7. Set break point.
8. Exit from DOS prompt.
9. Stop

Program:

```
.model tiny
.stack 32h
.code
org 2000h
start:      mov ax,cs
            mov ds,ax
            mov ax,00
            mov al,num1
            mov bl,num2
            sub al,bl
            mov result,al
            int 3
            mov ah,4ch
            int 21h
org 20f0h
            num1 db 0ffh
            num2 db 0aah
            result db 00
end start
```

Result:

Without borrow:

Input	Data	output	data
num1(20f0h)	0ffh	Result(20f2h)	055h
num2(20f1h)	0aah	carry	0

With borrow:

Input	Data	output	data
num1(20f0h)	0aah	Result(20f2h)	0abh
num2(20f1h)	0ffh	carry	1

8 bit subtraction of two hexadecimal numbers has been performed.

Exercise:

1. What AL has been used and not AX?
2. What happens if num1 contains 0AAH and num2 contains 0FFH?
3. How do you account for the difference obtained in previous question?

1.d - 16 BIT SUBTRACTION

Aim: To perform 16 bit subtraction of two hexadecimal numbers without and with borrow using input memory offset address 20f0h and the result stored at output memory offset address 20f4h.

Apparatus: TASM**, PC**

Algorithm:

1. Start
2. Initialize the address of code segment register to data segment register to use tiny memory model.
3. Load the first number to accumulator register from given memory location.
4. Load the second number to base register from given memory location.
5. Perform subtraction on first and second operands and store the result in destination operand.
6. Store the accumulator content to given memory location.
7. Set break point.
8. Exit from DOS prompt.
9. Stop

Program:

```
. model tiny
. stack 32h
. code
org 2000h
start:      mov ax,cs
            mov ds,ax
            mov ax,00h
            mov ax,num1
            mov bx,num2
            sub ax,bx
            mov result,ax
            int 3h
            mov ah,4ch
            int 21h
org 20f0h
            num1 dw 0ffffh
            num2 dw 0eabch
            result dw 00
end start
```

Result:

Without borrow:

Input	Data	output	data
num1(20f0h)	0ffffh	Result(20f4h)	1543h
num2(20f2h)	0eabch	carry	0

With borrow:

Input	Data	output	data
num1(20f0h)	0eabch	Result(20f4h)	0eabdh
num2(20f2h)	0ffffh	carry	1

16 bit subtraction of two hexadecimal numbers has been performed.

Exercise:

1. Why should AX be used not AL?
2. What happens if num1 and num2 values are interchanged?
3. If carry is set to 1 before subtraction what is the instruction to be used?

1.e – 8 BIT MULTIPLICATION

Aim: To perform 8 bit multiplication of two hexadecimal numbers using input memory offset address is 20f0h and the result is stored at output memory offset address is 20f2h.

Apparatus: TASM, PC

Algorithm:

1. Start
2. Initialize the address of code segment register to data segment register to use tiny memory model.
3. Load the first number to lower byte of accumulator register from given memory location.
4. Load the second number to lower byte of base register from given memory location.
5. Multiply first and second operands and store the result in destination operand.
6. Store the accumulator content to given memory location.
7. Set break point.
8. Exit from DOS prompt.
9. Stop

Program:

```
. model tiny
. stack 32h
. code
org 2000h
start:      mov ax,cs
            mov ds,ax
            mov ax,00
            mov al,num1
            mov bl,num2
            mul bl
            mov result,al
            mov result1,ah
            int 3
            mov ah,4ch
            int 21h
org 20f0h
            num1 db 0ffh
            num2 db 0aah
            result db 00
            result1 db 00
end start
```

Result:

Input	Data	output	data
num1(20f0h)	0ffh	Result(20f2h)	56h
num2(20f1h)	0aah	Result1(20f3h)	0a9h

8 bit multiplication of two hexadecimal numbers has been performed.

Exercise:

1. What is an extended accumulator?
2. AL and BL are used for multiplying why not AX & BX?
3. Instead of using MOV BL is it not possible to MUL num2?
4. What is the instruction used for signed multiplication?

1.f -16 BIT MULTIPLICATION

Aim: To perform 16 bit multiplication of two hexadecimal numbers using input memory offset address is 20f0h and the result is stored at output memory offset address is 20f4h.

Apparatus: TASM, PC

Algorithm:

1. Start
2. Initialize the address of code segment register to data segment register to use tiny memory model.
3. Load the first number to accumulator register from given memory location.
4. Load the second number to base register from given memory location.
5. Multiply first and second operands and store the result in destination operand.
6. Store the accumulator content to given memory location.
7. Store the data register content to the given memory location.
8. Set break point.
9. Exit from DOS prompt.
10. Stop

Program:

```
. model tiny
. stack 32h
. code
org 2000h
start:      mov ax,cs
            mov ds,ax
            mov ax,00
            mov dx, 00
            mov ax,num1
            mov bx,num2
            mul bx
            mov result,ax
            mov result1,dx
            int 3
            mov ah,4ch
            int 21h
org 20f0h
            num1 dw 0ffffh
            num2 dw 0aaaah
            result dw 00
            result1 dw 00
end start
```

Result:

Input	Data	output	data
num1(20f0h)	0ffffh	Result(20f4h)	5556h
num2(20f2h)	0aaaah	Result1(20f6h)	0aaa9h

16 bit multiplication of two hexadecimal numbers has been performed.

Exercise:

1. Why AL & BL are not used in this?
2. If result exceeds 32 bit where is it stored?
3. What is the name given to the register combination DXAX?

1.g – 8 BIT DIVISION

Aim: To perform 8 bit division of two hexadecimal numbers using input memory offset address is 20f0h and the result is stored at output memory offset address is 20f2h.

Apparatus: TASM**,** PC

Algorithm:

1. Start
2. Initialize the address of code segment register to data segment register to use tiny memory model.
3. Load the first number to lower byte of accumulator register from given memory location.
4. Load the second number to lower byte of base register from given memory location.
5. Divide first and second operands and store the result in destination operand.
6. Store the accumulator content to given memory location.
7. Set break point.
8. Exit from DOS prompt.
9. Stop

Program:

```
. model tiny
. stack 32h
. code
org 2000h
start:      mov ax,cs
            mov ds,ax
            mov ax,00
            mov dx,00
            mov al,num1
            mov bl,num2
            div bl
            mov Quotient,al
            mov remainder,ah
            int 3
            mov ah,4ch
            int 21h
org 20f0h
            num1 db 0ffh
            num2 db 0aah
            Quotient db 00
            remainder db 00
end start
```

Result:

Input	Data	Output	Data
num1(20f0h)	0ffh	Quotient (20f2h)	01h
num2(20f1h)	0aah	remainder (20f3h)	55h

8 bit division of two hexadecimal numbers has been performed.

Exercise:

1. Why is the registers DX & AX made zero in the above program?
2. The above program?
3. Where is the remainder in 8 bit division?
4. Where is the quotient in 8 bit division?
5. If AH contains a non-zero value, what will be the result of the division?
6. Which interrupt is used when a divide overflow error occurs?

1.h - 16 BIT DIVISION

Aim: To perform 16 bit division of two hexadecimal numbers using input memory offset address is 20f0h and the result is stored at output memory offset address is 20f4h.

Apparatus: TASM, PC

Algorithm:

1. Start
2. Initialize the address of code segment register to data segment register to use tiny memory model.
3. Load the first number to accumulator register from given memory location.
4. Load the second number to base register from given memory location.
5. Divide first and second operands and store the result in destination operand.
6. Store the accumulator content to given memory location.
7. Store the data register content to given memory location.
8. Set break point.
9. Exit from DOS prompt.
10. Stop

Program:

```
.model tiny
.stack 32h
.code
org 2000h
start:      mov ax,cs
            mov ds,ax
            mov ax,00
            mov dx,00
            mov ax,num1
            mov bx,num2
            div bx
            mov Quotient,ax
            mov remainder,dx
            int 3
            mov ah,4ch
            int 21h
org 20f0h
            num1 dw 0ffffh
            num2 dw 0aaaah
            Quotient dw 00
            remainder dw 00
end start
```

Result:

Input	Data	output	data
num1(20f0h)	0ffffh	Quotient (20f4h)	0001h
num2(20f2h)	0aaaah	remainder (20f6h)	5555h

16 bit division of two hexadecimal numbers has been performed.

Exercise:

1. What happens if DX register contains a nonzero value before DIV instruction?
2. What is the instruction used for signed division?
3. In the above program instead of DIV BX is it possible to use DIV num2?

Lab Incharge **HOD**

2.a -SORTING 'N' NUMBERS IN ASCENDING ORDER

Aim: To sort N numbers in a given array as ascending order using input memory offset address is 300h and result is stored at output memory offset address is 300h.

Apparatus: TASM, PC

Algorithm:

Step I: Initialize the data segment memory.
Step II : Initialize the number of elements counter
Step III : Initialize the comparisons counter..
Step IV: Load the numbers into respective registers.
Step V: Compare the elements. If first element < second element goto step VII Else go to next step.
Step VI: Swap the numbers in the memory.
Step VII: Increment memory pointer & Decrement the comparison counter.
Step VIII: Is count = 0 ? if yes go to next step else go to step IV.
Step IX: decrement the element counter.
Step X: Is count not 0 ? go Step III else go to next step Step IX: Stop & terminate the program

Program:

```
. model tiny
. stack 32h
Data segment
org 300h
        array db 05h,03h,01h,04h,02h
        count db 05h
Data ends
Code segment
         assume cs: code,ds: data
        org  2000h
Start:  mov ax,data
        mov ds,ax
        sub ax,ax
        mov dl,count
        mov di,dx
        dec di
back1:dec dx
        mov cx,dx
        mov si,offset array
back:   mov al,[si]
        cmp al,[si+1]
```

```
        jbe forw
        xchg al,[si+1]
        mov [si],al
forw:inc si
        loop back
        dec di
        jnz back1
        int 3h
        mov ah,4ch
        int 21h
Code ends
End start
```

Result:

Input	Data	Output	Data
array(300h)	05h, 03h, 01h, 04h, 02h	300h	01h, 02h, 03h, 04h, 05h

The sorting of N numbers in a given array has been performed

Exercise:

1. What is the purpose served by the following register?
    a) di
    b) si
    c) cx
2. jbe is equivalent to what other conditional jump instruction?
3. What is the difference between XCHG & CMP?
4. What is the purpose served by the instruction SUB AX, AX?

2.b - SORTING 'N' NUMBERS IN DESCENDING ORDER

Aim: To sort Nnumbers in a given array as descending order using input memory offset address is 300h and result is stored at output memory offset address is 300h.

Apparatus: TASM**, PC**

Algorithm:

Step I: Initialize the data segment memory.
Step II : Initialize the number of elements counter
Step III : Initialize the comparisons counter..
Step IV: Load the numbers into respective registers.
Step V: Compare the elements. If first element >second element goto step VII Else go to next step.
Step VI: Swap the numbers in the memory.
Step VII: Increment memory pointer & decrement the comparison counter.
Step VIII: Is count = 0 ? if yes go to next step else go to step IV.
Step IX: decrement the element counter.
Step X: Is count not 0 ? go Step III else go to next step Step IX: Stop & terminate the program

Program:

```
.model tiny
.stack 32h
Data1 segment
org 300H
        array db 05h,03h,01h,04h,02h
        count db 05h
Data1 ends
Code1 segment
        assume cs:code1,ds:data1
        org 2000h
Start:  mov ax,data1
        mov ds,ax
        sub ax,ax
        mov dl,count
        mov di,dx
        dec di
back1: dec dx
        mov cx,dx
        mov si,offset array
back:   mov al,[si]
        cmp al,[si+1]
```

```
        jae  forw
        xchg al,[si+1]
        mov [si],al
forw:   inc si
        loop back
        dec di
        jnz back1
        int 3h
        mov ah,4ch
        int 21h
Code1 ends
End start
```

Result:

Input	Data	Output	Data
array(300h)	05h,03h,01h,04h,02h	300h	05h,04h,03h,02h,01h

The sorting of N numbers in a given array has been performed

Exercise:

1. What are the flags which are checked when JG is executed and their conditions?
2. If array of 16 numbers is to be sorted, then what will be change in the following instructions?
   CMP AL, [SI+1]
   XCHG AL, [SI+1]
   INC SI
3. Write comment on LOOP instruction?

Lab Incharge **HOD**

3.SEARCHING A CHARACTER

Aim: To find the searching of a character in a given string and corresponding resultant message is displayed at command window**.**

Apparatus:TASM**,** PC

Algorithm:

1. Start
2. Data Segment Initialization
3. Extra Segment Initialization
4. Copy offset address of string1 to DI Register
5. Store Character to be search in AL Register
6. Initialize DF =0
7. Load the count of string1 in CL register.
8. Scan the string1 with respect to AL content.
9. If match found print character found message and if match not found in the scanning process till end character of string, then print character not found message on command window.
10. Stop

Program:

```
.model tiny
data segment
        notfound db 'Character not found in string$'
        found db 'Wow!!! Character found in string$'
data ends
extra  segment
        string1 db 'Fools can ask question which clever cannot answer',24H
        strlen equ ($-string1)
extra  ends
code segment
      assume cs:code,ds:data,es:extra
start:  mov ax,data
                mov ds,ax
        mov ax,extra
        mov es,ax
        mov di,offset string1
        mov al,'q'
        cld
                mov cx,strlen
        repne scasb
        jz foundchar
                mov ah,09h
        mov dx,offset notfound
                int 21h
```

```
                jmp exitp
foundchar:      mov ah,09h
        mov dx,offset found
                int 21h
exitp:          int 3h
                mov ah,4ch
                int 21h
code ends
end start
```

Result:

The searching of a character in a given string has been performed.

Exercise:

1. How is the search character given to the program ?
2. What registers are used by the instruction SCASB ?
3. What does the instruction REPNE do in this program.
4. What are the other variations of REP? Explain each of them.

Lab Incharge **HOD**

4.a -BLOCK TRANSFER

Aim: To perform transfer of a block from data segment to extra segment using string instructions.

Apparatus:TASM**,** PC

Algorithm:

1. Start
2. Initialize Data Segment
3. Initialize Extra Segment
4. Initialize SI with source offset address and DI with destination offset address.
5. Initialize DF =0
6. Store counts value of block in CX register.
7. Copy a content (byte) from data segment in extra segment till count reaches zero.
8. Set Break point Interrupt.
9. Exit from DOS prompt.
10. Stop

Program:

```
. model tiny
data segment
        srcdata db 'Empty vessels make much noise',24h
data ends
extra segment
        dstdata db 29 dup(0)
extra ends
code segment
        assume cs:code,ds:data,es:extra
start:          mov ax,data
                mov ds,ax
                mov ax,extra
                mov es,ax
                mov si,offset srcdata
                mov di,offset dstdata
                cld
                mov cx,29
                rep movsb
                nop
                int 3h
                mov ah,4ch
                int 21h
code ends
end start
```

Result:

Transfer of a block from data segment to extra segment has been performed.

Exercise:

1. If the DF=1, will the SI and DI register decrement?
2. The destination memory is pointed by which register combination?
3. The source is pointed to by which register combination?

4.b - STRING REVERSAL

Aim: To perform reversal of a given string is available at data segment and store the resultant string at extra segment using string instructions.

Apparatus: TASM, PC

Algorithm:

1. Start
2. Initialize Data Segment
3. Initialize Extra Segment
4. Initialize SI with source offset address of string1 and DI with destination offset address of string2.
5. Perform addition on DI register and length of the string and store in DI register.
6. Initialize DF =0
7. Store counter register with length of the string.
8. Copy a content (byte) from data segment to extra segment till count reaches zero.
9. Set break point Interrupt.
10. Exit from DOS prompt.
11. Stop

Program:

```
.model tiny
data segment
        string1 db 'Empty'
        strlen equ ($-string1)
data ends
extra segment
        string2  db   5 dup(0)
extra ends
code segment
        assume cs:code,ds:data,es:extra
start:  mov ax,data
        mov ds,ax
        mov ax,extra
        mov es,ax
        mov bx,offset string1
        mov si,bx
        mov di,offset string2
        add  di,5
        cld
        mov cx,strlen
back:   mov al,[si]
        mov es:[di],al          ; Segment override prefix
```

```
        inc  si
        dec di
        loop back
        int 3h
        mov ah,4ch
        int 21h
code ends
end start
```

Result:

The reversal of string has been performed.

Exercise:

1. Why BX register is added with ‘5’?
2. Why MOVS instruction is not used?
3. What is the function of LODS and STOS instructions?

4.c -STRING INSERTION

Aim: To perform insertion of sub string in to the main string is available at data segment and the resultant string is stored at extra segment using string instructions.

Apparatus:TASM**,** PC

Algorithm:

1. Start
2. Initialize Data Segment
3. Initialize Extra Segment
4. Store SI with source offset address of string1 and DI with destination offset address of string2.
5. Initialize DF =0
6. Move desired count value of string to CL register.
7. Copy Byte by Byte from data segment to extra segment till count reached to zero.
8. Read a byte from standard keyboard and write in the extra segment till the desired sub string is stored into the extra segment based on DL byte resister.
9. Remaining data segment string content copy to extra segment based on desired count.
10. Set Break point Interrupt.
11. Exit from DOS prompt.
12. Stop

Program:

```
.model tiny
data segment
        string1 db 'Empty vessels more noise$'
        strlen equ ($-string1)
data ends
extra segment
        string2 db strlen+5 dup(0)
extra ends
code segment
        assume cs:code,ds:data,es:extra

start:  mov ax, data
                mov ds, ax
                mov ax, extra
                mov es, ax
                mov si,offset string1
                mov di,offset string2
                cld
                mov cx,14
                rep movsb
                mov dl,5
```

```
back:   mov ah,01
        int 21h
        stos string2
        dec dl
        jnz back
        mov cx,11
        rep movsb
        nop
        int 3h
        mov ah,4ch
        int 21h
code ends
end start
```

Result:

The string insertion has been performed.

Exercise:

1. Why register 'DI' is loaded with 5?
2. What is the function of rep movsb?
3. What is the purpose of mov ah,01h & int 21h?

4.d - STRING DELETION

Aim: To perform deletion of sub string from the main string is available at data segment and the resultant string is stored at extra segment using string instructions.

Apparatus:TASM, PC

Algorithm:

1. Start
2. Initialize Data Segment
3. Initialize Extra Segment
4. Store SI with source offset address of Data Segment and DI with offset address of extra segment.
5. Initialize DF =0
6. Copy content from data segment to extra segment based on desired count value of CX register reaches to zero.
7. Add SI with desired value to copy from desired value.
8. Initialize desired count value in CX register.
9. Copy a content Byte by Byte from data segment to extra segment till count reaches to zero
10. Set Break point Interrupt.
11. Exit from DOS prompt.
12. Stop

Program:

```
.model tiny
data segment
        string1 db 'Empty vessels make more noise$'
        strlen equ ($-string1)
data ends
extra segment
        string2 db strlen-5 dup(0)
extra ends
code segment
        assume cs:code,ds:data,es:extra
start:          mov ax,data
                mov ds,ax
                mov ax,extra
                mov es,ax
                mov si,offset string1
                mov di,offset string2
                cld
                mov cx,13
                rep movsb
```

```
        cld
        mov si,18
        mov cx,12
        rep movsb
        int 3h
        mov ah,4ch
        int 21h
code ends
end start
```

Result:

The string deletion has been performed.

Exercise:

1. What is the purpose of string length?
2. What does ‘equ’ stand for?
3. What is the purpose of label start after the end directive?

4.e - LENGTH OF THE STRING

Aim: To find the length of the string is available at data segment and the correspondent result display at command window using string instructions.

Apparatus:TASM,PC

Algorithm:

1. Start
2. Initialize Data Segment
3. Copy the length of the string to BL register and initialize to zero.
4. Store SI with source offset address of string1.
5. Load a value from data segment string to AL and increment CL register.
6. Compare the content of AL register and end character of string.
7. If CF=1 go to step 5 other wise move CL value to res variable then compare CL with BL register, if both are equal then print string length found correct message otherwise print string length found incorrect on command window.
8. Set Break point Interrupt.
9. Exit from DOS prompt.
10. Stop

Program:

```
.model tiny
data segment
        string1 db 'Empty vessels make more noise$'
        strlen equ ($-string1)
        res db 0
        cort db  'strlength found correct', 0ah,0dh,24h
        incort db 'strlength found incorrect',  0ah,0dh,24h
data ends
code segment
        assume cs:code,ds:data
start:      mov ax,data
            mov ds,ax
            sub cl,cl
            mov bl,strlen
            mov si,offset string1
back:       lodsb
            inc cl
            cmp al,'$'
            jnz back
            mov res,cl
            cmp cl,bl
            jz correct
            mov dx,offset incort
```

```
                mov ah,09
                int 21h
                jmp exit
correct:        mov dx,offset cort
                mov ah,09
                int 21h
exit:           int 3h
                mov ah,4ch
                int 21h
code ends
end start
```

Result:

Thelength of the string has been performed.

Exercise:

1. What is the operation performed by the instruction cmp al, '$'?
2. What is function 09h / int 21h performed?
3. Why SI has not been incremented is the program?

4.f -STRING COMPARISION

Aim: To perform comparison of two strings, a source string is available at data segment and another string is available at extra segment then the correspondent resultant message is display at command window.

Apparatus:TASM**,** PC

Algorithm:

1. Start
2. Initialize Data Segment
3. Initialize Extra Segment
4. Store SI with source offset address of string1 and DI with offset address of string2.
5. Initialize DF =0
6. Store CL with length of the string.
7. Compare byte to byte from data segment counters with extra segment contents ill count reaches to zero.
8. If two strings are equal, then print strings are equal message otherwise print strings are not equal on command window.
9. Set Break point Interrupt.
10. Exit from DOS prompt.
11. Stop

Program:

```
.model tiny
data segment
        string1 db ‘Empty'
        strlen equ ($-string1)
        notsful db 'strings are unequal$'
        sful db 'strings are equal$'
data ends
extra segment
        string2 db‘Empty'
extra ends
code segment
        assume cs:code,ds:data,es:extra
start:      mov ax,data
            mov ds,ax
            mov ax,extra
            mov es,ax
            mov si,offset string1
            mov di,offset string2
            cld
            mov cx,strlen
```

```
                rep cmpsb
                jz forw
                mov ah,09h
                mov dx,offset notsful
                int 21h
                jmp exitp
forw:           mov ah,09h
                mov dx,offset sful
                int 21h
exitp:          int 3h
                mov ah,4ch
                int 21h
code ends
end start
```

Result:

Thecomparison of the two strings has been performed

Exercise:

1. What is the significance of CLD?
2. How does CMPSB perform the comparison?
3. Write comment on REP instruction?

Lab Incharge **HOD**

5.ARITHMETIC , LOGICAL AND BIT MANIPULATION OPERATIONS IN 8051

Aim: To perform arithmetic, logical and bit manipulation operations of 8051 microcontroller.

Apparatus: Keil IDE, PC

Algorithm:

1. Start
2. Move desired values or first number in accumulator register immediately.
3. Move desired values or second number in b register immediately.
4. Perform addition on the contents of a and b registers and store result in a register.
5. Perform subtraction on the contents of a and b registers and store results in a register.
6. Perform Multiplication on the contents of a and b registers and store result in a register.
7. Perform division on the contents of a and b registers and store result in a register.
8. Perform OR, AND, XOR, NOT operations and bit manipulations on a and b registers and check the resultant.
9. Stop

Program:

```
start:
        mov a, #34h          ;addition
        mov b, #44h
        add a, b
        mov a, #44h          ; subtraction
        mov b,#34h
        subb a, b
        mov a, #34h          ; multiplication
        mov b, #44h
        mul ab
        mov a, #50h          ; division
        mov b, #05h
        div ab
        mov a, #05h          ; or operation
        anl a, #6
        mov a, #03h          ;and operation
        orl a, #05
        mov a, #05h          ; rotate right
        rr a
        mov a, #05h          ; rotate left
        rl a
        clr p1.0             ; bit manipulation
```

```
        setb p1.0
        clr c                   ; bit manipulation
        setb c
end
```

Result:

The arithmetic, logical and bit manipulation operations have been performed.

Exercise:

1. Write arithmetic operations?
2. Write comment on MUL AB?
3.Write comment on DIV AB?
4.Define Immediate addressing mode?

Lab Incharge **HOD**

6.TIMER COUNTER IN 8051

Aim: To perform Timer0 as Timer in mode 0 operation.

Apparatus: Keil IDE, Prog Isp, PC, 8051 micro controller kit, parallel bus, USB powered 89S52 Programmer.

Theory:

TMOD Bit Function:

D7	D6	D5	D4	D3	D2	D1	D0
Gate	C/$\bar{T}$	M1	M0	Gate	C/$\bar{T}$	M1	M0

Gate - OR gate enable bit which controls RUN/STOP of timer 1. Set to 1 by program to enable timer to run if bit TR1 in TCON is set and signal on external interrupt INT1 is high. Cleared to 0 by program to enable time to run if bit TR1 is set

C/$\bar{T}$- Set to 1 by program to make timer 1 act as a counter by counting pulses from external input pins 3.5. Cleared to zero by program to make timer act as a timer by counting internal frequency.

M1 - Timer/counter operating mode select bit 1. Set/cleared by program to select mode.

M0 - Timer/counter operating mode select bit 0. Set/cleared by program to select mode.

Gate -OR gate enable bit which controls RUN/STOP of timer 0. Set to 1 by program to enable timer to run if bit TR0 in TCON is set and signal on external interrupt INT0 is high. Cleared to 0 by program to enable time to run if bit TR0 is set.

C/$\bar{T}$- Set to 1 by program to make timer 0 act as a counter by counting pulses from external input pin 3.4. Cleared to zero by program to make timer act as a timer by counting internal frequency.

M1 - Timer/counter operating mode select bit 1. Set/cleared by program to select mode

M0 - Timer/counter operating mode select bit 0. Set/cleared by program to select mode.

M1	M0	Mode	Description
0	0	0	Use the THX register as an 8-bit counter and the TLX as a 5-bit counter.
0	1	1	Use the THX register as an 8-bit counter and the TLX as an 8-bit counter.
1	0	2	Use only the TLX register as an 8-bit counter.
1	1	3	In modes 0 - 2, Timers 0 and 1 may be programmed independently. In mode 3: Timer 0 in mode 3 becomes two separate 8-bit counters.

Timer 1 in mode 3 may still be used, but will generate no interrupts.

TMOD is not bit addressable. Direct Byte Address is 89h. The only difference in counting and time is the source of the clock pulses to the counters.

TCON bit function:

8051 Timer Control

D7	D6	D5	D4	D3	D2	D1	D0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

(TCON) Special Function Register

Bit	Symbol	TCON Bit Function
7	TF1	Timer 1 Overflow flag. Set when timer rolls from all 1's to 0. Cleared when processor vectors to execute interrupt service routine located at program address 001Bh.
6	TR1	Timer 1 run control bit. Set to 1 by program to enable timer to count; cleared to 0 by program to halt timer.
5	TF0	Timer 0 Overflow flag. Set when timer rolls from all 1's to 0. Cleared when processor vectors to execute interrupt service routine located at program address 000Bh.
4	TR0	Timer 0 run control bit. Set to 1 by program to enable timer to count; cleared to 0 by program to halt timer.
3	IE1	External interrupt 1 Edge flag. Set to 1 when a high-to-low edge signal is received on port 3.3 (INT1). Cleared when processor vectors to interrupt service routine at program address 0013h. Not related to timer operations.
2	IT1	External interrupt 1 signal type control bit. Set to 1 by program to enable external interrupt 1 to be triggered by a falling edge signal. Set to 0 by program to enable a low-level signal on external interrupt 1 to generate an interrupt.
1	IE0	External interrupt 0 Edge flag. Set to 1 when a high-to-low edge signal is received on port 3.2 (INT0). Cleared when processor vectors to interrupt service routine at program address 0003h. Not related to timer operations.
0	IT0	External interrupt 0 signal type control bit. Set to 1 by program to enable external interrupt 1 to be triggered by a falling edge signal. Set to 0 by program to enable a low-level signal on external interrupt 0 to generate an interrupt.

Bit addressable as TCON.0 to TCON.7, Direct Byte Address is 88h.

Interrupt Enable:

D7	D6	D5	D4	D3	D2	D1	D0
EA	X	ET2	ES	ET1	EX1	ET0	EX0

Address: 0A8H (bit addressable)
EA – Global interrupt enable
X – Not defined
ET2 – Timer 2 interrupt enable
ES – Serial port interrupt enable
ET1 – Timer 1 interrupt enable
EX1 – External interrupt 1 enable
ET0 – Timer 0 interrupt enable
EX0 – External interrupt 0 enable

Algorithm:

1. Start
2. Initialize timer0 interrupt enable using IE Register.
3. Initially store timer0 with Zero
4. Initialize r7 as counter but initially zero.
5. Start timer0
6. Go to ISR of timer0 if timer0 interrupt occurs using its vector address else wait.
7. Jump to step 3.
8. Stop timer0 in ISR, increment r7 by 1 on each interrupt then copy content of r7 register to port 0
9. Delay
10. Move T0 with zero and start timer then return to step5
11. Stop

Program:

```
                ORG 0000
LJMP   START
                ORG 0003H
INTRO:
                ORG 000BH
TIMER0:
LJMP TIMER0INTR
                ORG 0013H
INTR1:
                ORG 001BH
TIMER1:
                ORG 0023H
SERrx_tx:
```

```
                    ORG   40H
start:
        mov     psw,#00
        mov sp,#60h
        mov ie,#10000010b ; enable all - ex0,tm0,ex1,tm1,ser
        mov  th0,#00
        mov   tl0,#00
        mov r7,#0
        setb  tr0
        jmp $

org   200h
timer0intr:
        clr tr0
         inc r7
        mov p0,r7
        call delay
        mov  th0,#0       ;reload the higher byte
        mov tl0,#0        ; reload the lower byte of timer so timing
                          ; restarts with desired value
        setb tr0          ;start the timer operations
        reti              ; return from the interrupt
delay:
        mov r3,#0ffh
h2:     mov r4,#0ffh
h1:     djnz  r4,h1
        djnz r3,h2
        ret
end
```

Result:

The Timer as timer in mode 0 operation has been performed.

Exercise:

1. Write TMOD format?
2. Write TCON format?
3. Write comment JMP $

Lab Incharge **HOD**

7. INTERRUPT HANDLING IN 8051

Aim: To perform interrupt handling in 8051 microcontroller**.**

ApparatusKeil IDE, Prog Isp, PC, 8051 micro controller kit, parallel bus, USB powered 89S52 Programmer

Theory:

Interrupt EnableRegister:

D7	D6	D5	D4	D3	D2	D1	D0
EA	X	ET2	ES	ET1	EX1	ET0	EX0

Address: 0A8H (bit addressable)
EA – Global interrupt enable
X – not defined
ET2 – Timer 2 interrupt enable
ES – Serial port interrupt enable
ET1 – Timer 1 interrupt enable
EX1 – External interrupt 1 enable
ET0 – Timer 0 interrupt enable
EX0 – External interrupt 0 enable

Interrupt Priority Register :

D7	D6	D5	D4	D3	D2	D1	D0
-	-	PT2	PS	PT1	PX1	PT0	PX0

Address: 0B8H (bit addressable)
x – not defined
PT2 – Priority for timer 2 interrupt
PS – Priority for serial port interrupt
PT1 – Priority for timer 1 interrupt
PX1 – Priority for external interrupt 1
PT0 – Priority for timer 0 interrupt
PX0 – Priority for external interrupt 0

Algorithm:

1. Start
2. Configure intr0, intr1 interrupt enable using IE Register.
3. Initialize r7 as counter but initially zero.
4. Start checking interrupts.
5. Go to ISR of intr0 if intr0 interrupt occurs using its vector address else go to step 8

6. In ISR of intr0, increment r7 by 1 on each interrupt then copy content of r7 register to port 0.
7. Give delay.
8. Go to ISR of intr1, if intr1 interrupt occurs using its vector address else go to step 4
9. In ISR of intr1, decrement r7 by 1 on each interrupt then copy content of r7 register to port 0.
10. Go to step 4.
11. Stop

Program:

```
                        ORG 0000
LJMP   START
                        ORG 0003H
EXTERNALINTRO:
LJMP  EXINTR0
                        ORG 000BH
TIMER0:
                        ORG 0013H
EXTERNALINTR1:
LJMP  EXINTR1
                        ORG 001BH
TIMER1:
                        ORG 0023H
SERrx_tx:

                        ORG   40H
start:
        mov     psw,#00
        mov sp,#60h
        mov p0,#0h
        mov p3,#0fh
        clr it0
setb  it0 ; make the interrupt0 on transition high to low
        setb  it1; make the interrupt1 on transition high to low
        mov ie,#10000101b ; enable all - ex0 & ex1
        mov r7,#0
         jmp $
        org   200h
exintr0:
        inc r7
        mov p0,r7
        call delay
        reti                    ; return from the interrupt
```

```
exintr1:
        dec r7
        mov p0,r7
        call delay
        reti

delay:
        mov r3,#0ffh
h2:     mov r4,#0ffh
h1:     djnz  r4,h1
        djnz r3,h2
        ret
end
```

Result:

The interrupt handling of 8051 operation has been performed.

Exercise:

1. Write IE format?
2. Write differences between edge and level triggering?
3. Write comment on DJNZ instruction?
4. Write comment on RETI instruction?

Lab Incharge **HOD**

8.UART Operation in 8051

Aim: To perform serial communication between 8051 kit and PC using UART of 8051.

Apparatus: Keil IDE, Prog Isp, PC, 8051 micro controller kit, Rs 232 cableUSB powered 89S52 Programmer.

Theory:

One of the 8051s many powerful features is its integrated UART, otherwise known as a serial port. The fact that the 8051 has an integrated serial port means that you may very easily read and write values to the serial port. If it were not for the integrated serial port, writing a byte to a serial line would be a rather tedious process requiring turning on and off one of the I/O lines in rapid succession to properly "clock out" each individual bit, including start bits, stop bits, and parity bits. However, we do not have to do this. Instead, we simply need to configure the serial ports operation mode and baud rate. Once configured, all we have to do is write to an SFR to write a value to the serial port or read the same SFR to read a value from the serial port. The 8051 will automatically let us know when it has finished sending the character we wrote and will also let us know whenever it has received a byte so that we can process it. We do not have to worry about transmission at the bit level--which saves us quite a bit of coding and processing time.

SCON:

Setting the Serial Port Mode

The first thing we must do when using the 8051s integrated serial port is, obviously, configure it. This lets us tell the 8051 how many data bits we want, the baud rate we will be using, and how the baud rate will be determined.

First, let's present the "Serial Control" (SCON) SFR and define what each bit of the SFR represents:

Bit	Name	Bit Address	Explanation of function
0	RI	98h	Receive Flag. Set when a byte has been completely received.
1	TI	99h	Transmit Flag. Set when a byte has been completely transmitted.
2	RB8	9Ah	Receive bit 8. The 9th bit received in mode 2 and 3.
3	TB8	9Bh	Transmit bit 8. The 9th bit to transmit in mode 2 and 3.
4	REN	9Ch	Receiver Enable. This bit must be set in order to receive characters.
5	SM2	9Dh	Multiprocessors Communications Enable (explained later)
6	SM1	9Eh	Serial port mode bit 1.
7	SM0	9Fh	Serial port mode bit 0

Additionally, it is necessary to define the function of SM0 and SM1 by an additional table:

SM0	SM1	Serial Mode	Explanation	Baud Rate
0	0	0	8-bit Shift Register	Oscillator / 12
0	1	1	8-bit UART	Set by Timer 1 (*)
1	0	2	9-bit UART	Oscillator / 64 (*)
1	1	3	9-bit UART	Set by Timer 1 (*)

(*) Note: The baud rate indicated in this table is doubled if PCON.7 (SMOD) is set.

The SCON SFR allows us to configure the Serial Port. Thus, well go through each bit and review its function.

The first four bits (bits 4 through 7) are configuration bits.

Bits **SM0** and **SM1** let us set the serial mode to a value between 0 and 3, inclusive. The four modes are defined in the chart immediately above. As you can see, selecting the Serial Mode selects the mode of operation (8-bit/9-bit, UART or Shift Register) and also determines how the baud rate will be calculated. In modes 0 and 2 the baud rate is fixed based on the oscillator's frequency. In modes 1 and 3 the baud rate is variable based on how often Timer 1 overflows. We talk more about the various Serial Modes in a moment.

The next bit, **SM2**, is a flag for "Multiprocessor communication." Generally, whenever a byte has been received the 8051 will set the "RI" (Receive Interrupt) flag. This lets the program know that a byte has been received and that it needs to be processed. However, when SM2 is set the "RI" flag will only be triggered if the 9th bit received was a "1". That is to say, if SM2 is set and a byte is received whose 9th bit is clear, the RI flag will never be set. This can be useful in certain advanced serial applications. For now it is safe to say that you will almost always want to clear this bit so that the flag is set upon reception of *any* character.

The next bit, **REN**, is "Receiver Enable." This bit is very straightforward: If you want to receive data via the serial port, set this bit. You will almost always want to set this bit.The last four bits (bits 0 through 3) are operational bits. They are used when actually sending and receiving data--they are not used to configure the serial port.

The **TB8** bit is used in modes 2 and 3. In modes 2 and 3, a total of nine data bits are transmitted. The first 8 data bits are the 8 bits of the main value, and the ninth bit is taken from TB8. If TB8 is set and a value is written to the serial port, the datas bits will be written to the serial line followed by a "set" ninth bit. If TB8 is clear the ninth bit will be "clear."

The **RB8** also operates in modes 2 and 3 and functions essentially the same way as TB8, but on the reception side. When a byte is received in modes 2 or 3, a total of nine bits are received. In this case, the first eight bits received are the data of the serial byte received and the value of the ninth bit received will be placed in RB8.

TI means "Transmit Interrupt." When a program writes a value to the serial port, a certain amount of time will pass before the individual bits of the byte are "clocked out" the serial port. If the program were to write another byte to the serial port before the first byte was completely output, the data being sent would be garbled. Thus, the 8051 lets the program know that it has "clocked out" the last byte by setting the TI bit. When the TI bit is set, the program may assume that the serial port is "free" and ready to send the next byte.

Finally, the **RI** bit means "Receive Interrupt." It functions similarly to the "TI" bit, but it indicates that a byte has been received. That is to say, whenever the 8051 has received a complete byte it will trigger the RI bit to let the program know that it needs to read the value quickly, before another byte is read.

Setting the Serial Port Baud Rate

Once the Serial Port Mode has been configured, as explained above, the program must configure the serial ports baud rate. This only applies to Serial Port modes 1 and 3. The Baud Rate is determined based on the oscillator's frequency when in mode 0 and 2. In mode 0, the baud rate is always the oscillator frequency divided by 12. This means if your crystal is 11.059 MHz, mode 0 baud rate will always be 921,583 baud. In mode 2 the baud rate is always the oscillator frequency divided by 64, so a 11.059Mhz crystal speed will yield a baud rate of 172,797.

In modes 1 and 3, the baud rate is determined by how frequently timer 1 overflows. The more frequently timer 1 overflows, the higher the baud rate. There are many ways one can cause timer 1 to overflow at a rate that determines a baud rate, but the most common method is to put timer 1 in 8-bit auto-reload mode (timer mode 2) and set a reload value (TH1) that causes Timer 1 to overflow at a frequency appropriate to generate a baud rate.

To determine the value that must be placed in TH1 to generate a given baud rate, we may use the following equation (assuming PCON.7 is clear).

TH1 = 256 - ((Crystal / 384) / Baud)

If PCON.7 is set then the baud rate is effectively doubled, thus the equation becomes:

TH1 = 256 - ((Crystal / 192) / Baud)

For example, if we have an 11.059 MHz crystal and we want to configure the serial port to 19,200 baud we try plugging it in the first equation:

TH1 = 256 - ((Crystal / 384) / Baud)
TH1 = 256 - ((11059000 / 384) / 19200)
TH1 = 256 - ((28,799) / 19200)
TH1 = 256 - 1.5 = 254.5

As you can see, to obtain 19,200 baud on a 11.059Mhz crystal we have to set TH1 to 254.5. If we set it to 254 we will have achieved 14,400 baud and if we set it to 255 we will have achieved 28,800 baud. Thus were stuck...

But not quite... to achieve 19,200 baud we simply need to set PCON.7 (SMOD). When we do this we double the baud rate and utilize the second equation mentioned above. Thus we have:

TH1 = 256 - ((Crystal / 192) / Baud)
TH1 = 256 - ((11059000 / 192) / 19200)
TH1 = 256 - ((57699) / 19200)
TH1 = 256 - 3 = 253

Here we are able to calculate a nice, even TH1 value. Therefore, to obtain 19,200 baud with an 11.059MHz crystal we must:

1. Configure Serial Port mode 1 or 3.
2. Configure Timer 1 to timer mode 2 (8-bit auto-reload).
3. Set TH1 to 253 to reflect the correct frequency for 19,200 baud.
4. Set PCON.7 (SMOD) to double the baud rate.

Writing to the Serial Port

Once the Serial Port has been property configured as explained above, the serial port is ready to be used to send data and receive data. If you thought that configuring the serial port was simple, using the serial port will be a breeze.To write a byte to the serial port one must simply write the value to the **SBUF** (99h) SFR. For example, if you wanted to send the letter "A" to the serial port, it could be accomplished as easily as:

```
MOV SBUF, #A
```

Upon execution of the above instruction the 8051 will begin transmitting the character via the serial port. Obviously transmission is not instantaneous--it takes a measureable amount of time to transmit. And since the 8051 does not have a serial output buffer we need to be sure that a character is completely transmitted before we try to transmit the next character.

The 8051 lets us know when it is done transmitting a character by setting the **TI** bit in SCON. When this bit is set we know that the last character has been transmitted and that we may send the next character, if any. Consider the following code segment:

```
CLR TI; be sure the bit is initially clear
MOV SBUF, #A; Send the letter A to the serial port
JNB TI, $; Pause until the TI bit is set.
```

The above three instructions will successfully transmit a character and wait for the TI bit to be set before continuing. The last instruction says "Jump if the TI bit is not set to $"--$, in most assemblers, means "the same address of the current instruction." Thus the 8051 will pause on the JNB instruction until the TI bit is set by the 8051 upon successful transmission of the character.

Reading the Serial Port

Reading data received by the serial port is equally easy. To read a byte from the serial port one just needs to read the value stored in the**SBUF** (99h) SFR after the 8051 has automatically set the **RI** flag in SCON.

For example, if your program wants to wait for a character to be received and subsequently read it into the Accumulator, the following code segment may be used:

JNB RI,$;Wait for the 8051 to set the RI flag
MOV A,SBUF ;Read the character from the serial port

The first line of the above code segment waits for the 8051 to set the RI flag; again, the 8051 sets the RI flag automatically when it receives a character via the serial port. So as long as the bit is not set the program repeats the "JNB" instruction continuously.

Once the RI bit is set upon character reception the above condition automatically fails and program flow falls through to the "MOV" instruction which reads the value.

Algorithm:

1. Start
2. Initialize TMOD register with 20H immediately to operate timer1 in mode2 as a timer.
3. Move TH1 register with FDH to get the 9600 baud rate on serial communication.
4. Initialize SCON register with 52 H to configure serial port mode operation.
5. Start timer1.
6. Initialize new line and carriage for serial communication.
7. Store DPTR register a with offset address of text.
8. Call subroutine to print the desired message.
9. Call get char subroutine to read character from the keyboard serially.
10. Call put char subroutine to write a character on serial port.
11. Repeat 9

Program:

```
start:
        mov   tmod,#00100000b
        mov   th1,#0fdh
        setb  tr1
        mov   scon,#01010010b

        call newline
        mov   dptr,#txt
        call  putstring
repeat: call getchar
        call put char
        anl a,#0fh
```

```
        mov p0 ,a
        sjmp  repeat
putstring:
        clr   a
        movc  a,@a+dptr
        jz    exit
        call  putchar
        inc   dptr
        sjmp  putstring
exit:   ret
putchar:
        jnb  ti,$
        clr  ti
        mov  sbuf,a
        ret
getchar:
        jnb  ri,$
        clr  ri
        mov a,sbuf
        ret
newline:
        mov a,#0dh
        call putchar
        mov a,#0ah
        call putchar
        ret
txt:    db    'mpmc lab', 0ah,0dh,00h
        end
```

Result:

The serial communication between 8051 kit and PC operation has been performed.

Exercise:

1. Write SCONformat?
2. Write function of SBUF?
3. Write comment on JNB TI, $?

Lab Incharge **HOD**

9. Stepper with 8051

Aim: To perform stepper motor Interfacing with 8051.

Apparatus: Keil IDE, Prog Isp, PC, 8051 micro controller kit, parallel bus, stepper motor, USB powered 89S52 Programmer

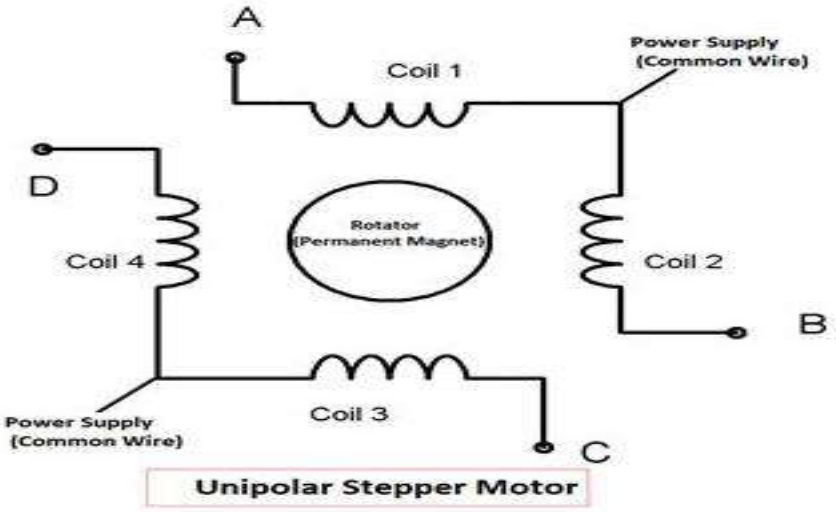
Theory:

Introduction

Data acquisition and control represents the most popular applications of microprocessors. Stepper Motor control is a very popular applications of microprocessors in control area, as stepper motor is capable of accepting pulses directly from the microprocessor and move accordingly.

There are three types of stepper motors:

a) Permanent magnet (PM)]
b) Variable Reluctance (VR)
c) Hybrid Synchronous Stepper Motor



Specification of the stepper motor used:

The motor is reversible one with a torque of 3kgcm. The power requirement is +5VDC @1.2A current per winding at full torque. The step angle is 1.8^o, i.e for every single excitation, the motor shaft rotates by 1.8^o. for the motor to rotate one full revolution (360^o), number of steps required is 200. The stepper motor used has four stator windings which are brought out through colored wires terminated at a 4 pin connector.

Stepping Angle = 360/ No. of rotor teeth

Where stepper motor no. of rotor teeth are 200, hence stepping angle is 1.8°

Clock wise Rotation

Steps	A	B	C	D
1	0	1	1	1
2	1	0	1	1
3	1	1	0	1
4	1	1	1	0

Anticlock wise Rotation

Steps	A	B	C	D
1	1	1	1	0
2	1	1	0	1
3	1	0	1	1
4	0	1	1	1

Stepper motor connections - PIN Number from bottom on kit

Brown (Ground)-2PIN
Red -4PIN
Orange -5 PIN
Yellow -6 PIN
Green -7 PIN

Algorithm:

1. Start
2. Configure port0 as output port.
3. Move p0 with 70 immediately then call delay subroutine.
4. Move p0 with b0 immediately then call delay subroutine.
5. Move p0 with d0 immediately then call delay subroutine.
6. Move p0 with e0 immediately then call delay subroutine.
7. Go to step 3.
8. Stop

Program:

```
org 00h
mov p0, #00h // initiates p0 as the output port
main:  mov p0, #70h
       call delay
       call delay
       mov p0, #0b0h
       call delay
       call delay
       mov p0, #0d0h
       call delay
       call delay
```

```
        mov p0, #0e0h
        call delay
        call delay
    sjmp main ; jumps back to the main program
delay:
        mov r3, #0ffh
h2:     mov r4, #0ffh
h1:     djnz  r4, h1
        djnz r3, h2
        ret
end
```

Result:

The Stepper motor interfacing with 8051 has been performed.

Exercise:

1. How many steps are there for revolutions?
2. Each step corresponds to how many degrees?
3. How many coils dose the stepper motors have?
4. What is the purpose of resistor connected between base & ground?

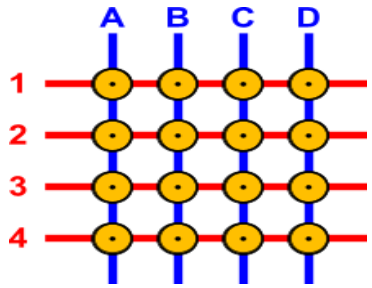
10. Matrix Keyboard 8051 kit and PC

Aim: To perform matrix keyboard interfacing using 8051 microcontroller trainer kit and PC.

Apparatus: Keil IDE, Prog Isp, PC, 8051 micro controller kit, parallel bus, USB powered 89S52 Programmer

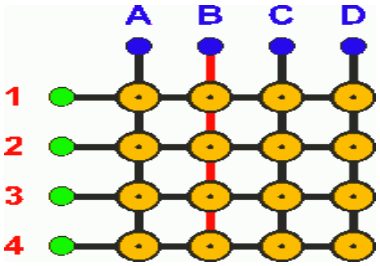
Theory:

The matrices are actually an interface technique. It can be used to interface inputs like the PC keyboard keys, but also to control multiple outputs like LEDs. According to this technique, the I/O are divided into two sections: the columns and the rows. You can imagine a matrix as an excel sheet. Here is a 4 x 4 matrix.



The blue lines are the columns and the red lines the rows. There are 16 knots that the rows and columns intersect. The columns and the rows are NOT in contact! Suppose that we want to make a key matrix. To do this, we will have to connect a button to each knot. The buttons will have a push-to-make contact. When the operator pushes this button, it will connect the column and the row that it corresponds to. Now I will put the push-to-make buttons onto the matrix. The buttons are named with the Column:Row name that they connect. For example, the top-left button is named A1 and the bottom right is named D4.

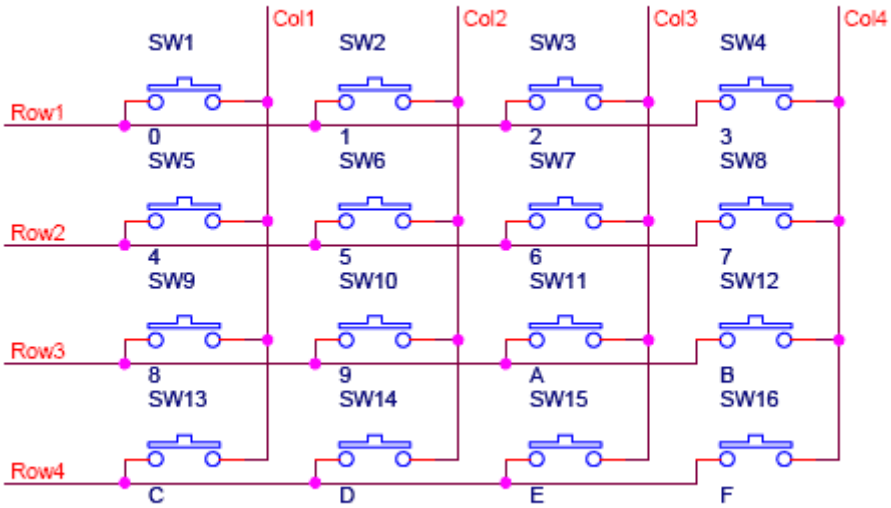
Key-matrix working



To understand the operation principle, we will re-draw the above matrix without colors. we will also put connection pins to each row and column wire. Then, we will give power to only one column, the column B. The wire that is red, indicates that it has power, and the button that is purple indicates that the button is pressed. Then, we will simulate a button press to button number B3:

Watch the above animation. The column wire B has power all the time. No other wire has power, until the button B3 is pressed. This button makes contact between the column B and the row 3. Because column B has power, the row 3 will also have power as long as the button B3 is pressed! What this means it that, if we know which column has currently power, and we watch the rows, then we can understand which button was pressed, if we detect power on a row! If for example we know that the column B has power, and we detect also power to row 3, then we understand that the button B3 is pressed.

Keyboard Circuit Diagram



Algorithm:

1. Start
2. Call a subroutine to initialize LCD module interface with 8051.
3. Print predefined message on LCD module.
4. Configure P1 upper lines as input port and P1 lower lines as output port.
5. Scan the key row 0 to row3 then check the button pressing status from coloumn0 to coloum3 then if key has been pressed then send its corresponding ASCII value send to LCD module else go to next step.
6. Repeat 5
7. Stop

Program:

```
start:
        call lcm
lcm:    call lcm_init
        clr p3.5
        call again
        jmp $
w_msg:
```

```
        mov a,#0
        movc a,@a+dptr
        cjne a,#00,do_msg
        ret
do_msg:
        call w_data
        inc dptr
        jmp  w_msg
w_data:
        setb     p3.6 ;          //register select
        setb p3.7 ;              //enable
        mov p0,a;
        clr p3.7;                //enable
        clr p3.6 ;               //register select
        call delay15;
        ret
w_instrn:
        clr p3.6;                //register select
        clr p3.7 ;               //enable
        mov p0,a ;
        setb p3.7 ;      //enable
        clr p3.7 ;               //enable
        call delay15 ;
        call delay15
        ret
delay15:
        mov r7,#255
        djnz  r7,$
        ret
lcm_init:
        call delay15;
        mov a,#30h
        call w_instrn  ;
        mov a,#30h
        call w_instrn;
        mov a,#30h
        call w_instrn;
        mov a,#38h
        call w_instrn;
        mov a,#0fh
        call w_instrn;   w_instrn(0x38);
        mov a,#01h
        call w_instrn;   w_instrn(0x38);
        mov a,#06h
```

```
        call w_instrn;   w_instrn(0x38);
         ret
again:  mov a,#'k'
        call w_data
        mov a,#'e'
        call w_data
        mov a,#'y'
        call w_data
key :   mov p1,#0f0h
         clr p1.0
         setb p1.1
        setb p1.2
         setb p1.3
k0:     jb p1.4, k1
        call delay
        call delay
        mov a, #30h
        call w_data
        call delay
        call delay
 k1:    jb p1.5, k2
        call delay
        call delay
        mov a, #31h
        call w_data
        call delay
        call delay
k2:     jb p1.6, k3
        call delay
        call delay
        mov a, #32h
        call w_data
        call delay
         call delay
k3:     jb p1.7, k4
        call delay
        call delay
        mov a, #33h
        call w_data
        call delay
        call delay
k4:     setb p1.0
        clr p1.1
        jb p1.4, k5
```

```
call delay
        call delay
mov a, #34h
call w_data
call delay
call delay
 k5:jb p1.5, k6
call delay
call delay
        mov a, #35h
        call w_data
        call delay
        call delay
k6:     jb p1.6, k7
        call delay
        call delay
        mov a, #36h
        call w_data
        call delay
        call delay
k7:     jb p1.7, k8
        call delay
        call delay
        mov a, #37h
        call w_data
        call delay
        call delay
k8:     setb p1.1
        clr p1.2
        jb p1.4, k9
        call delay
        call delay
        mov a, #38h
        call w_data
        call delay
        call delay
 k9:    jb p1.5, k10
        call delay
        call delay
        mov a, #39h
        call w_data
        call delay
k10:    jb p1.6, k11
        call delay
```

```
        call delay
        mov a, #41h
        call w_data
        call delay
        call delay
k11:    jb p1.7, k12
        call delay
        call delay
        mov a, #42h
        call w_data
        call delay
        call delay
k12:    setb p1.2
        clr p1.3
        jb p1.4, k13
        call delay
        call delay
        mov a, #43h
        call delay
        call delay
 k13:   jb p1.5, k14
        call delay
        call delay
        mov a, #44h
call w_data
call delay
call delay
k14:    jb p1.6, k15
        call delay
        call delay
        mov a, #45h
        call w_data
        call delay
        call delay
k15:    jb p1.7, k16
        call delay
        call delay
        mov a, #46h
        call w_data
call delay
call delay
k16:    setb p1.3
jmp key
delay : mov  r6,#0a6h
```

```
dd1:    mov r7,#50h
        djnz  r7,$
        djnz  r6,dd1
        ret
        end
```

Result:

The matrix keyboard interfacing with 8051 has been performed.

Exercise:

1. Write SCON format?
2. Write matrix keyboard scanning?
3. Define debounce delay

Lab Incharge **HOD**

11.Seven Segment Display (SSD) interface with 8051 kit

Aim: To perform seven segment display(SSD) interfacing using 8051 microcontroller trainer kit.

Apparatus: Keil IDE, Prog Isp, PC, 8051 micro controller kit, USB powered 89S52 Programmer

Theory:

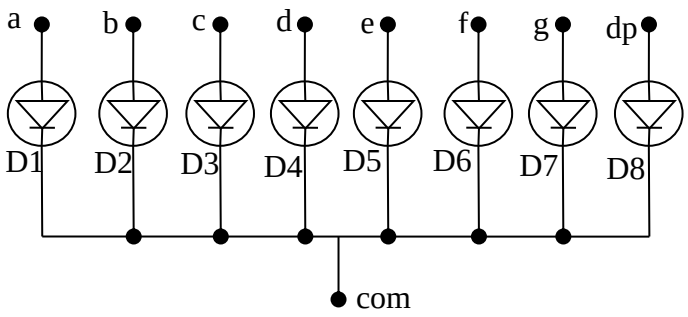
Seven segment displays are used to indicate numerical information. Seven segments display can display digits from 0 to 9 and even we can display few characters like A, b, C, H, E, e, F, etc. These are very popular and have many more applications. 7 Segment Display works by interfacing 7 Segment Display to 8051 Microcontroller.

This system displays the digits from 0 to 9 continuously with a predefined delay.

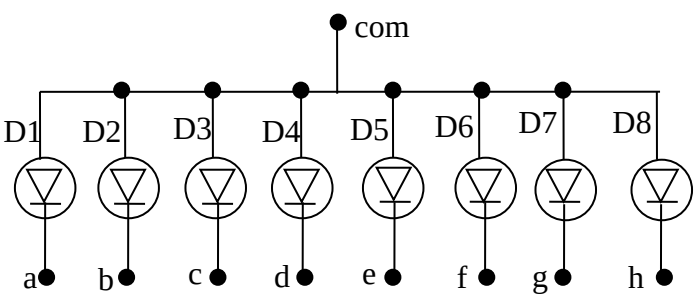
Circuit Principle:

Seven segment displays internally consist of 8 LEDs. In these LEDs, 7 LEDs are used to indicate the digits 0 to 9 and single LED is used for indicating decimal point. Generally seven segments are two types, one is common cathode and the other is common anode.

In common cathode, all the cathodes of LEDs are tied together and labeled as common and the anode are left alone. In common anode, seven segments display all the anodes are tied together and cathodes are left freely. Below figure shows the internal connections of seven segment Display.

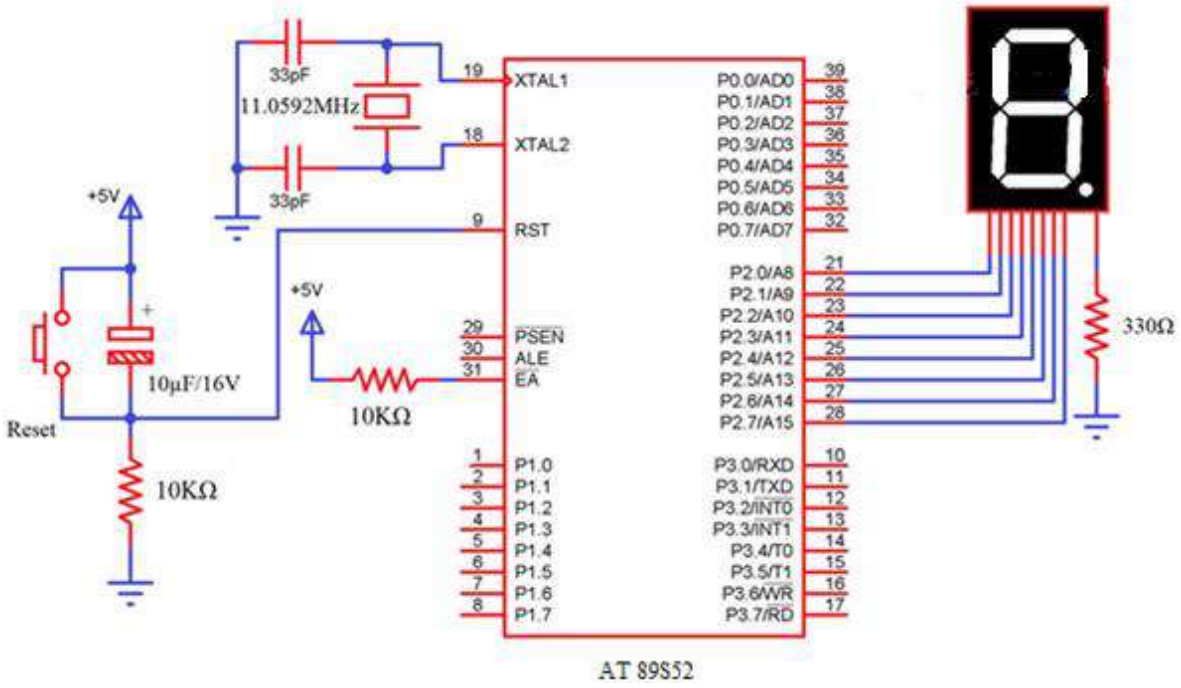


Common Cathode

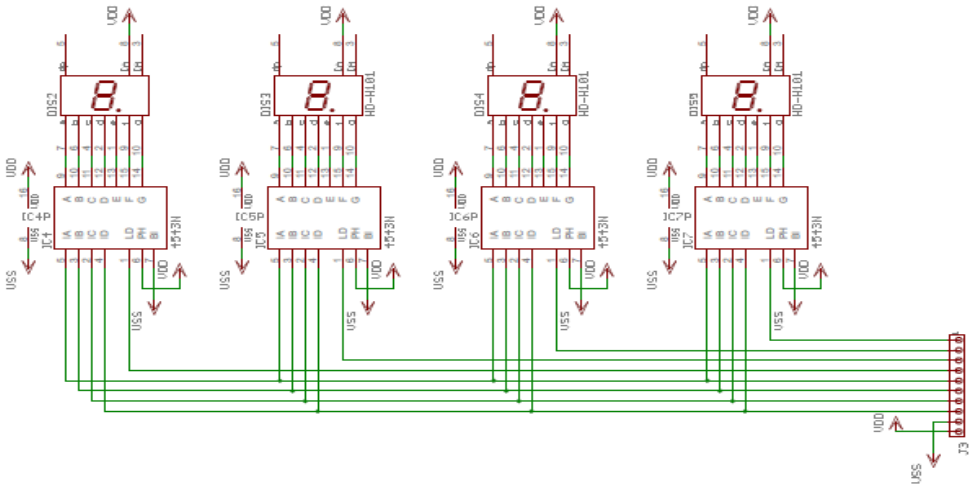


Common Anode

Circuit Diagram



Schematic Diagram :



Algorithm:

1. Start
2. Configure P2 as output port.
3. Move accumulator with 01H immediately then send the content of accumulator to P2 to display as 0 on SSD devices and give delay.

4. Select SSD device by writing logic1 to P2.0 then use P2 upper lines to print digits on SSD devices from 1 to 9 (BCD logic representation) and give delay on each content to display on SSD devices respectively.
5. Go to step 3
6. Stop

Program:

```
org 0000h
start:      mov p2,#00h
            setb p2.0
            l1:  mov a,#01h
            mov p2,a
            call delay
            call delay
            call delay
            mov a,#11h
            mov p2,a
            call delay
            call delay
            call delay
            mov a,#21h
            mov p2,a
            call delay
            call delay
            call delay
            mov a,#31h
            mov p2,a
            call delay
            call delay
            call delay
            mov a,#41h
            mov p2,a
            call delay
            call delay
            call delay
            mov a,#51h
            mov p2,a
            call delay
            call delay
            call delay
            mov a,#61h
```

```
        mov p2,a
        call delay
        call delay
        call delay
        mov a,#71h
        mov p2,a
        call delay
        call delay
        call delay
        mov a,#81h
        mov p2,a
        call delay
        call delay
        call delay
        mov a,#91h
        mov p2,a
        call delay
        call delay
        call delay
        sjmp l1

delay:
        mov r3,#0ffh
h2:     mov r4,#0ffh
h1:     djnz r4,h1
        djnz r3,h2
        ret
end
```

Result:

The SSD interfacing with 8051 has been performed.

Exercise:

1. Write types of seven segment displays.
2. How decimal point is different from seven segments?
3. Write applications of seven segment display.

Lab Incharge **HOD**

12.LED interface with 8051 kit

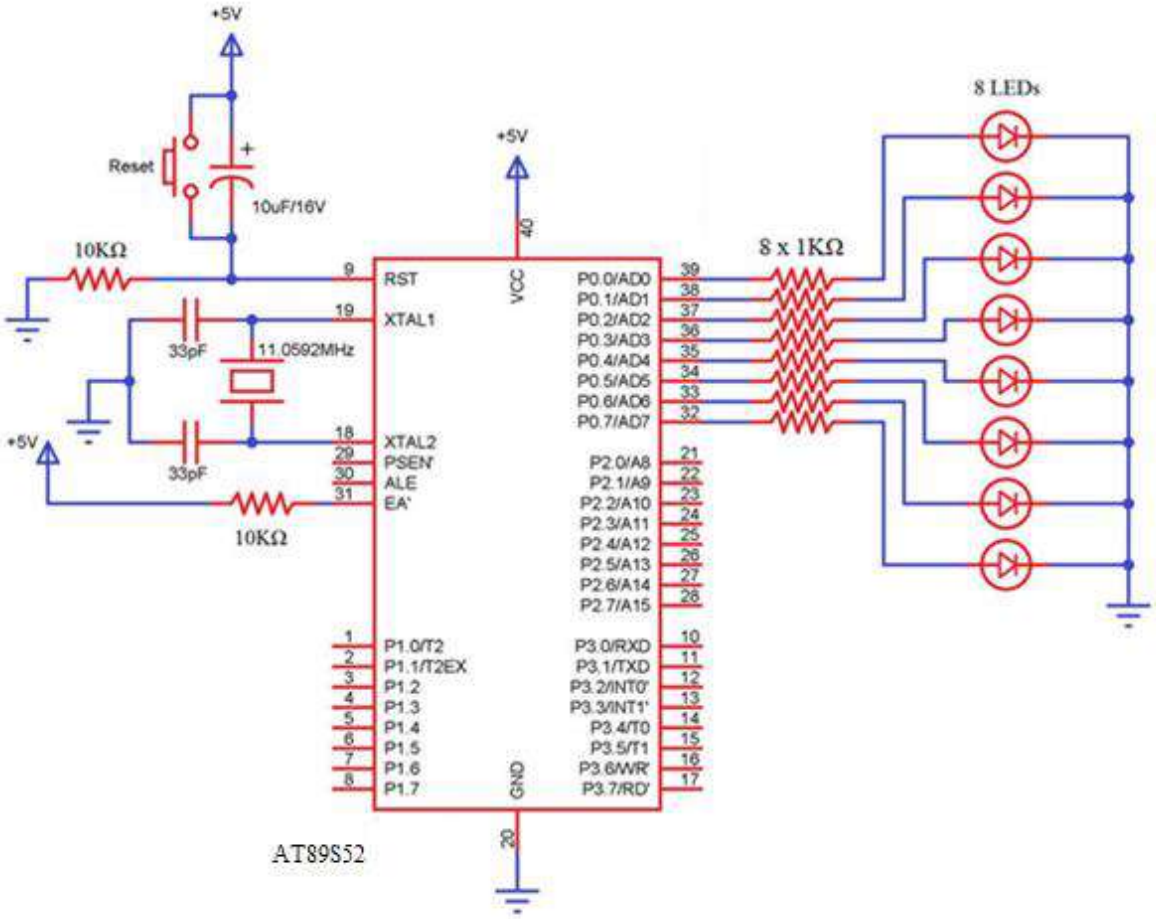
Aim: To perform LED interfacing using 8051 microcontroller trainer kit.

Apparatus: Keil IDE, Prog Isp, PC, 8051 micro controller kit, parallel bus, led display,USB powered 89S52 Programmer,

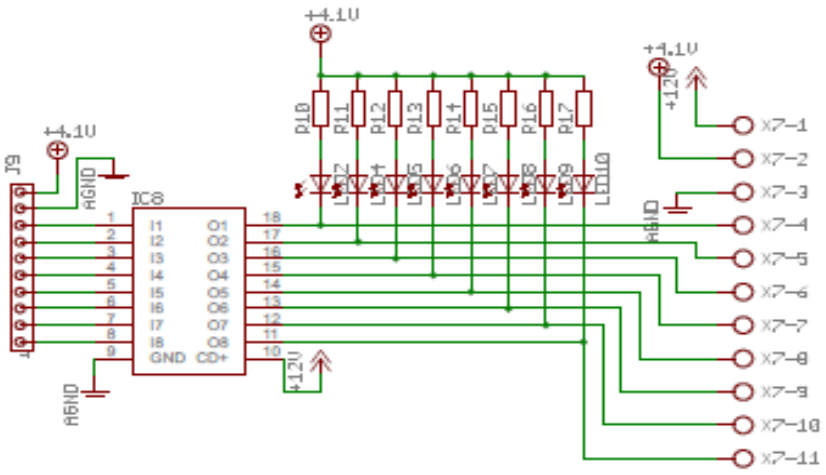
Theory:

The circuit mainly consists of AT89S52 microcontroller. AT89S52 belongs to the family of 8051 microcontroller. It is an 8-bit microcontroller. This microcontroller has 4KB of Flash Programmable and Erasable Read Only Memory and 128 bytes of RAM. It has two 16 bit timers/counters. It supports USART communication protocol. It has 40 pins. There are four ports are designated as P0, P1, P2, and P3. Here we used P0 as outputport.

Circuit Diagram :



Schematic Diagram :



Algorithm:

1. Start
2. Configure port0 as output port.
3. Move accumulator to zero immediately.
4. Move content of accumulator to port0
5. Give delay.
6. Move accumulator with FF immediately.
7. Move content of accumulator to port0
8. Give delay.
9. Go to step 3
10. Stop

Program :

```
org 0000h

start:
        mov a,00h
mov p0,a
        call delay
        call delay
        call delay
        mov a,0ffh
mov p0,a
        call delay
        call delay
        call delay
```

```
        sjmp start

delay:
        mov r3, #0ffh
h2:     mov r4, #0ffh
h1:     djnz r4, h1
        djnz r3, h2
ret

    end
```

Result :

The LED interfacing with 8051 has been performed.

Exercise:

1. Write abbreviation of LED.
2. Describe spontaneous emission of light in LED.
3. Which biasing is used in LED while turn on.

13. LCD Module Interfacing

Aim: To perform LCD module interfacing with 8051 micro controller.

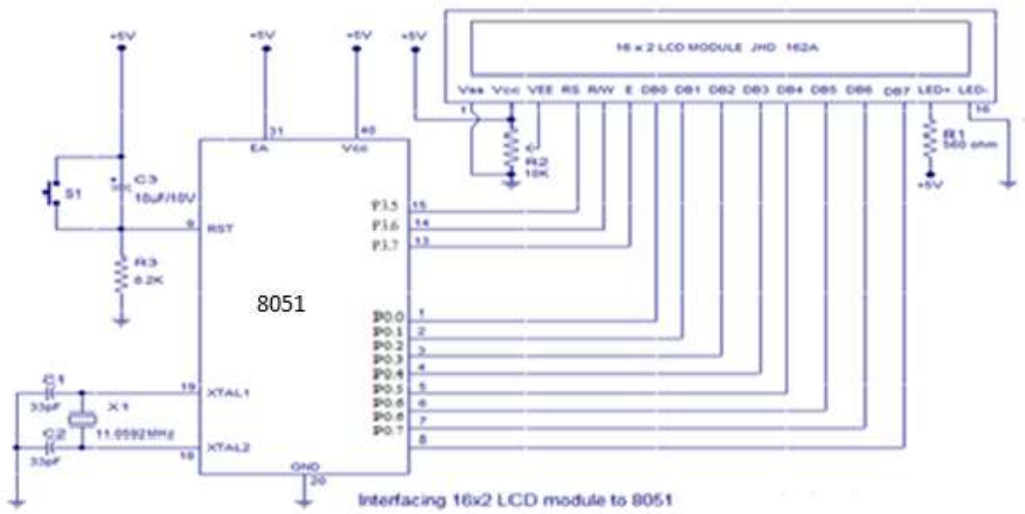
Apparatus: Keil IDE, Prog Isp, PC, 8051 micro controller kit, parallel bus, led display,USB powered 89S52 Programmer

Theory:

The Pin Assignment of LCD Module

Pin	Descriptiion
1	VSS GND(Ground)
2	Vcc (Supply Voltage)
3	Vee (Contrast Voltage)
4	RS (Instruction/Register select)
5	R/W (Read/Write)
6	E (Clock)
7	D0 (Data 0)
8	D1 (Data 1)
9	D2 (Data 2)
10	D3 (Data 3)
11	D4 (Data 4)
12	D5 (Data 5)
13	D6 (Data 6)
14	D7 (Data 7)
15	A(Led+) Anode for LED Backlight
16	K(LED-) Cathode for LED Backlight

LCD MODULE CIRCUIT DIAGRAM



Algorithm:

1. Start
2. Call a subroutine to initialize LCD modules Interface with 8051.
3. Move DPTR register with offset address of message1.
4. Call message display subroutine to print the predefined message by using DPTR and use w_data subroutine to send data on LCD and w_instrn subroutine to send command on LCD module.
5. Go to step 3.
6. Stop

Program:

```
start:
        call lcm
lcm:    call lcm_init
        clr p3.5
        //mov a,#48h
        //call w_data
        mov dptr,#msg1
        call  w_msg
        jmp $
w_msg:
        mov a, #0
        movc a,@a+dptr
        cjne a,#00,do_msg
        ret
do_msg:
        call w_data
        inc dptr
        jmp w_msg
w_data:
        setb    p3.6 ;          //register select
        setb p3.7 ;             //enable
        mov p0,a;
        clr p3.7;               //enable
        clr p3.6 ;              //register select
        call delay15;
        ret
w_instrn:
        clr p3.6;               //register select
        clr p3.7 ;              //enable
        mov p0,a ;
        setb p3.7 ;             //enable
```

```
        clr p3.7 ;                  //enable
        call delay15 ;
        call delay15
        ret
delay15:
        mov r7,#255
        djnz  r7,$
        ret
lcm_init:
        call delay15;
        mov a,#30h
        call w_instrn  ;
        mov a,#30h
        call w_instrn;
        mov a,#30h
        call w_instrn;
        mov a,#38h
        call w_instrn;
        mov a,#0fh
        call w_instrn;   w_instrn(0x38);
        mov a,#01h
        call w_instrn;   w_instrn(0x38);
        mov a,#06h
        call w_instrn;   w_instrn(0x38);
         ret
        msg1: db “welcome", 00
        end
```

Result:

The LCD module is interfaced with 8051micro controller.

Exercise:

1. Write LCD control word format?
2. How many characters are displayed at each line of LCD?
3. Write significance of RS and EN?
4. Write significance of 38H?

Lab Incharge **HOD**

14. Sequence Generator

Aim: To generate a Sequenceusing UART of 8051 micro controller.

Apparatus: Keil IDE, Prog Isp, PC, 8051 micro controller kit, parallel bus, USB powered 89S52 Programmer.

Algorithm :

1. Start
2. Initialize TMOD register with 20H immediately to configure timer1 in mode 2 as timer.
3. Move TH1 register with FDH to get the 9600 baud rate on serial communication.
4. Initialize SCON register with 52 H to configure serial port mode operation.
5. Start timer1.
6. Initialize new line and carriage for serial communication.
7. Store DPTR a with offset address of text.
8. Call putstring subroutine to print the desired message.
9. Initialize r1 with 09 and r0 with 00h immediately.
10. Increment r0 by 1 and add to accumulator.
11. Perform addition 30H with accumulator content.
12. Call putstring subroutine to print the number on serial monitor using accumulator content.
13. Decrement r1 by 1 check r1 value is zero if not zero go to step 10 otherwise next line.
14. Repeat step 9.

Program :

```
org 00h
start:
        mov   tmod,#00100000b
        mov   th1,#0fdh
        setb  tr1
        mov   scon,#01010010b
        call newline
        mov   dptr,#txt
        call  putstring
repeat:
        mov a,#00h
 mov r1, #09h
        mov b,a
        mov r0,#00h
 l1:    inc r0
        add a,r0
```

```
        orl a,#30h
        call putchar
        call newline
        clr a
        djnz r1,l1
        clr a
        sjmp  repeat
putstring:
        clr   a
        movc  a,@a+dptr
        jz    exit
        call  putchar
        inc   dptr
        sjmp  putstring
exit:   ret

putchar:
        jnb  ti,$
        clr  ti
        mov  sbuf,a
        ret
getchar:
        jnb  ri,$
        clr  ri
        mov a,sbuf
        ret
newline:
        mov a,#0dh
        call putchar
        mov a,#0ah
        call putchar
        ret
        txt:      db    'sequence is to be printed:', 0ah,0dh,00h
end
```

Result :

The generation of a sequence using UART of 8051 micro controller is done.

Exercise :

1. Write PCON format?
2. Write comment on JNB RI, $?

Lab Incharge **HOD**

15. Interface 8bit ADC to 8051

Aim: To interface 8 bit ADC with 8051 micro controller.]

Apparatus: Keil IDE, Prog Isp, PC, 8051 micro controller kit, parallel bus, USB powered 89S52 Programmer, ADC0804.

Theory:

Description :

The ADC0804 series are versatile 8-Bit μP compatible general purpose ADC converters operate on single 5-Vsupply. These devices are treated as a memory location or I/O port to a micro-processor system withoutadditional interface logic. The outputs are Tri-state latched which facilitate interfacing to micro-processor controlbus. The converter is designed with a differential potentiometric ladder, a circuit equivalent of the 256R network.It contains analog switches sequenced by successive approximation logic. A functional diagram of the ADCconverter is shown in *Functional Block Diagram*. All of the package pinouts are shown and the major logic controlpaths are drawn in heavier weight lines. The differential analog voltage input has good common mode-rejectionand permits offsetting the analog zero-input voltage value. Moreover, the input reference voltage can be adjustedto allow encoding small analog voltage span to the full 8-bits resolution. To ensure start-up under all possibleconditions, an external WR pulse is required during the first power-up cycle.

Using a SAR logic the most significant bit is tested first and after 8 comparisons (64 clock cycles) a digital 8-bitbinary code (1111 1111 = full-scale) is transferred to an output latch and then an interrupt is asserted ($\overline{INTR}$makes a high-to-low transition). A conversion in process can be interrupted by issuing a second start command.The device may be operated in the free-running mode by connecting $\overline{INTR}$ to the $\overline{WR}$ input with $\overline{CS}$=0.

On the high-to-low transition of the $\overline{WR}$ input the internal SAR latches and the shift register stages are reset. Aslong as the $\overline{CS}$ input and $\overline{WR}$input remain low, the ADC will remain in a reset state. Conversion will start from 1to 8 clock periods after at least one of these inputs makes a low-to-high transition.

The converter is started by having $\overline{CS}$ and $\overline{WR}$ simultaneously low. This sets the start flip-flop (F/F) and theresulting "1" level resets the 8-bit shift register, resets the Interrupt ($INTR$) F/F and inputs a "1" to the D flop,F/F1, which is at the input end of the 8-bit shift register. Internal clock signals then transfer this "1" to the Qoutput of F/F1. The AND gate, G1, combines this "1" output with a clock signal to provide a reset signal to thestart F/F. If the set signal is no longer present (either $\overline{WR}$or $\overline{CS}$ is a "1") the start F/F is reset and the 8-bit shiftregister then can have the "1" clocked in, which starts the conversion process. If the set signal were to still bepresent, this reset pulse would have no effect (both outputs of the start F/F would momentarily be at a "1" level)and the 8-bit shift register would continue to be held in the reset mode. This logic therefore allows for wide $\overline{CS}$and $\overline{WR}$ signals and the converter will start after at least one of these signals returns high and the internal clocksagain provide a reset signal for the start F/F.

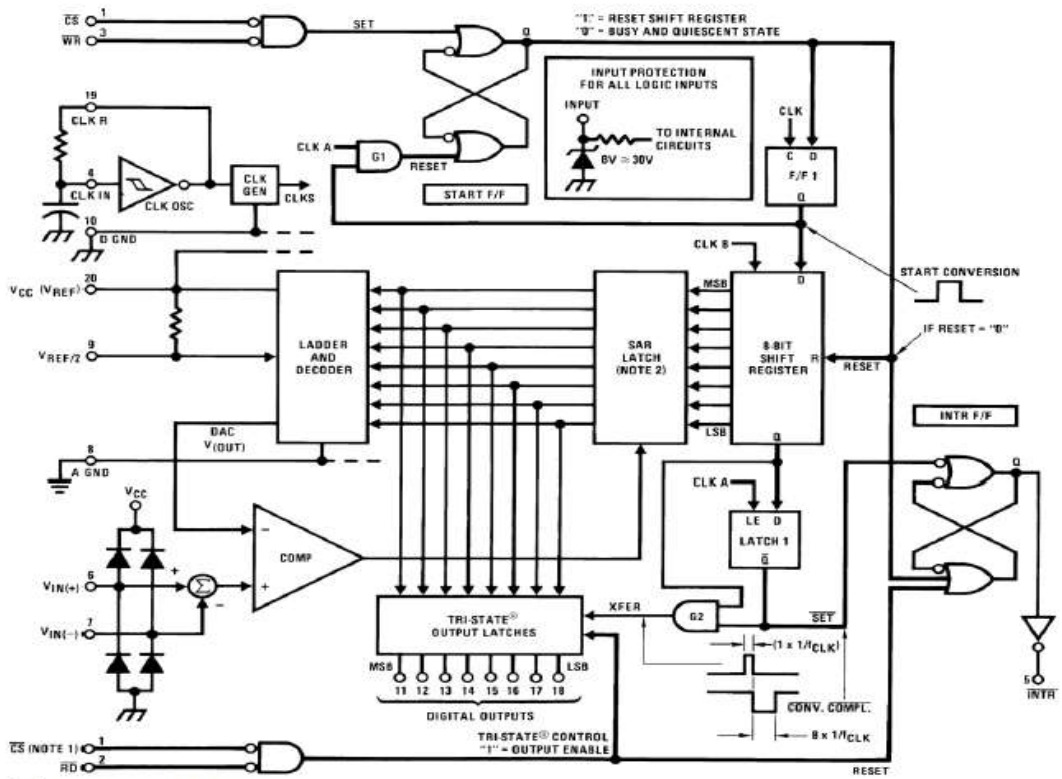
After the "1" is clocked through the 8-bit shift register (which completes the SAR search) it appears as the inputto the D-type latch, LATCH 1. As soon as this "1" is output from the shift register, the AND gate, G2, causes thenew digital word to transfer to the Tri-state output latches. When LATCH 1 is subsequently enabled, the $\overline{Q}$ outputmakes a high-to-low transition which causes the INTR F/F to set. An inverting buffer then supplies the $\overline{INTR}$ inputsignal.

Note this $\overline{SET}$ control of the $\overline{INTR}$ F/F remains low for 8 of the external clock periods (as the internal clocks runat 1/8 of the frequency of the external clock). If the data output is continuously enabled ($\overline{CS}$ and $\overline{RD}$ both heldlow), the $\overline{INTR}$ output will still signal the end of conversion (by a high-to-low transition), because the $\overline{SET}$ inputcan control the $\overline{Q}$ output of the INTR F/F even though the RESET input is constantly at a M "1M " level in thisoperating mode. This $\overline{INTR}$ output will therefore stay low for the duration of the $\overline{SET}$ signal, which is 8 periods ofthe external clock frequency (assuming the ADC is not started during this interval).

When operating in the free-running or continuous conversion mode ($\overline{INTR}$ pin tied to $\overline{WR}$ and $\overline{CS}$ wired low), the START F/F is SET by the high-to-low transition of the $\overline{INTR}$ signal. This resets theSHIFT REGISTER which causes the input to the D-type latch, LATCH 1, to go low. As the latch enable input isstill present, the $\overline{Q}$ output will go high, which then allows the $\overline{INTR}$ F/F to be RESET. This reduces the width ofthe resulting $\overline{INTR}$ output pulse to only a few propagation delays (approximately 300 ns).

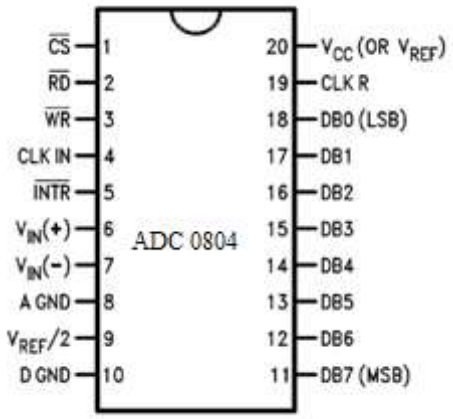
When data is to be read, the combination of both $\overline{CS}$ and $\overline{RD}$ being low will cause the $\overline{INTR}$ F/F to be reset andthe Tri-state output latches will be enabled to provide the 8-bit digital outputs.

Functional Block Diagram

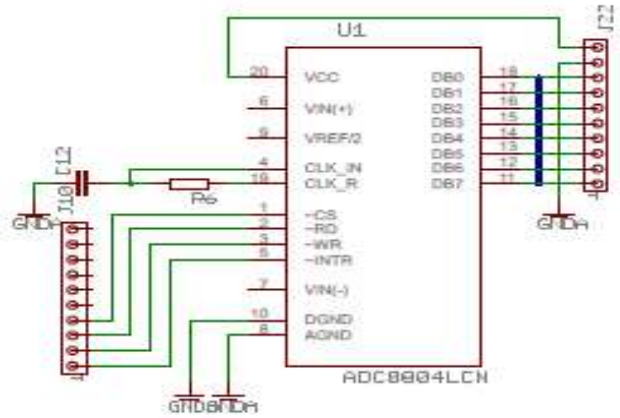


(1) $\overline{CS}$ shown twice for clarity.
(2) SAR = Successive Approximation Register.

ADC0804**Pin Configuration :**



Schematic Diagram:



Features:

Interfacing Logic Needed – Access Time 135 ns
Easy Interface to All Microprocessors/Microcontrollers, or Operates as a Stand-Alone Device
Differential Analog Voltage Inputs
Logic Inputs and Outputs Meet Both MOS and TTL Voltage-Level Specifications
Works with 2.5-V (LM336) Voltage Reference On-Chip Clock Generator
0-V to 5-V Analog Input Voltage Range with Single 5-V Supply
No Zero Adjust Required
0.3-Inch Standard Width 20-Pin DIP Package
20-Pin Molded Chip Carrier or Small Outline Package
Operates ratio metrically or with 5 VDC, 2.5 VDC, or Analog Span Adjusted Voltage Reference

Key Specifications
Resolution: 8 Bits
Total Error: ±1/4 LSB, ±1/2 LSB and ±1 LSB
Conversion Time: 100 μs

Applications:
Operates with Any 8-Bit μP Processors/Microcontroller or as a Stand-Alone Device
Interface to Temp Sensors, Voltage Sources, and Transducers

Pin Functions

PIN NO.	PIN NAME	I/O	DESCRIPTION
1	CS	I	Chip Select
2	RD	I	Read
3	WR	I	Write
4	CLK IN	I	External Clock input or use internal clock gen with external RC elements
5	INTR	O	Interrupt request
6	$V_{IN}(+)$	I	Differential analog input+
7	$V_{IN}(-)$	I	Differential analog input–
8	A GND	I	Analog ground pin
9	$V_{REF}/2$	I	Reference voltage input for adjustment to correct full scale reading
10	D GND	I	Digital ground pin
11	DB7	O	Data bit 7
12	DB6	O	Data bit 6
13	DB5	O	Data bit 5
14	DB4	O	Data bit 4
15	DB3	O	Data bit 3
16	DB2	O	Data bit 2
17	DB1	O	Data bit 1
18	DB0 (LSB)	O	Data bit 0
19	CLK R	I	RC timing resistor input pin for internal clock gen
20	V_{CC} (or V_{REF})	I	+5V supply voltage, also upper reference input to the ladder

AC Electrical Characteristics

The following specifications apply for VCC=5 VDC and TMIN≤ TA≤TMAX (unless otherwise specified)

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
T_C	Conversion Time	f_{CLK} = 640 kHz(1)	103		114	µs
		See (2)(1)	66		73	1/fCLK
f_{CLK}	Clock Frequency	V_{CC} = 5V(2)	100	640	1460	kHz
	Clock Duty Cycle		40%		60%	
CR	Conversion Rate in Free-Running Mode	INTR tied to $\overline{WR}$ with $\overline{CS}$ = 0 VDC, f_{CLK} = 640 kHz	8770		9708	conv/s

(1) Accuracy is specified at f_{CLK} = 640 kHz. At higher clock frequencies accuracy can degrade. For lower clock frequencies, the duty cycle limits can be extended so long as the minimum clock high time interval or minimum clock low time interval is no less than 275 ns.
(2) With an asynchronous start pulse, up to 8 clock periods may be required before the internal clock phases are proper to start the conversion process.

Algorithm:

1. Start
2.Initiates p1 as the input port
3. clear p3.3 to make cs=0
4. set p3.2 to make rd high
5. clear p3.1 to make wr low
6.set p3.1 high again to do low to high pulse to wr for starting conversion
7.polls until intr=0 using p3.0
8.clear p3.3 and p3.2 to ensures cs=0 and high to low pulse to rd for reading the data from ADC circuit.
9. moves the digital data to accumulator from p1
10. complements the digital data
11. rotate left 8 times with 1 bit wise in the accumulator content
12. send accumulator content to p0

13 Give delay
14.Go to step 2

Program :

```
org 00h
        mov p0,#00h
        mov p1,#11111111b // initiates p1 as the input port
main:   clr p3.3                    ;makes cs=0
        setb p3.2                   ; makes rd high
        clr p3.1                    ; makes wr low
        setb p3.1                   ; low to high pulse to wr for starting conversion
wait:   jb p3.0,wait                ; polls until intr=0
         clr p3.3                   ; ensures cs=0
        clr p3.2                     ; high to low pulse to rd for reading the data from adc
        mov a,p1                ; moves the digital data to accumulator
        cpl a     ; complements the digital data (*see the notes)
        rl a
        rl a
        rl a
        rl a
        rl a
        rl a
        rl a
        rl a
        mov p0,a                    ; outputs the data to p0 for the leds
        call delay
        sjmp main                    ; jumps back to the main program
delay:
        mov r3,#0ffh
h2:     mov r4,#0ffh
h1:     djnz r4,h1
        djnz r3,h2
        ret
end
```

Result:

Interfacing of 8 bitADC with 8051 has been performed.

Exercise:

1. ADC 0804 is based on …… circuitry.
2. What is the conversion time of ADC0804?
3. What is the operating clock frequency of ADC?

Lab Incharge **HOD**

16. Interface DAC to 8051

Aim: To generate triangular wave using DAC interface with 8051 micro controller.

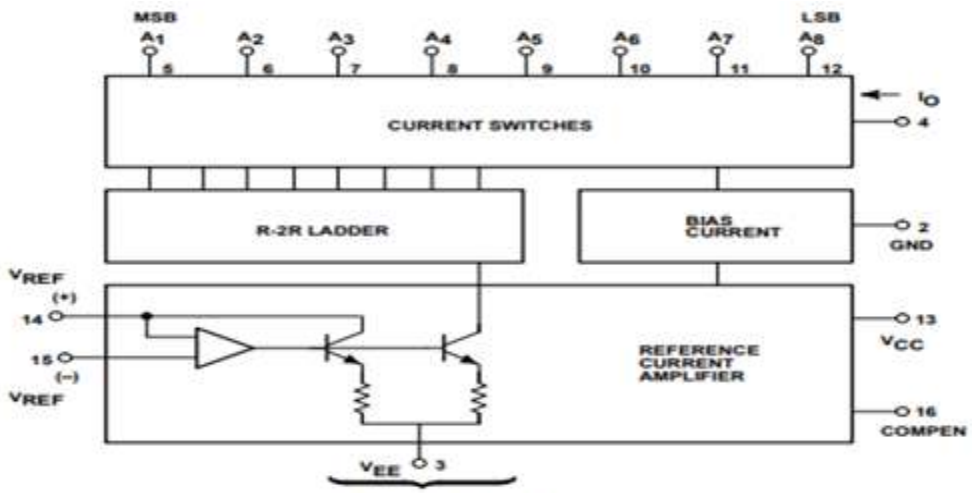
Apparatus: Keil IDE, Prog Isp, PC, 8051 micro controller kit, parallel bus, USB powered 89S52 Programmer, DCA1408.

Theory:

Circuit Description:

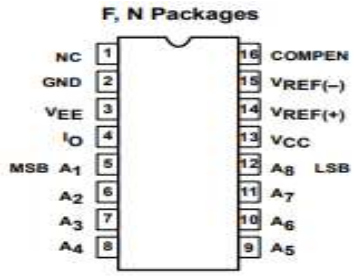
The MC1408 consists of a reference current amplifier, an R-2R ladder, and 8 high-speed current switches. For many applications, only a reference resistor and reference voltage need be added. The switches are non-inverting in operation; therefore, a high state on the input turns on the specified output current component. The switch uses current steering for high speed, and a termination amplifier consisting of an active load gain stage with unity gain feedback. The termination amplifier holds the parasitic capacitance of the ladder at a constant voltage during switching, and provides a low impedance termination of equal voltage for all legs of the ladder. The R-2R ladder divides the reference amplifier current into binarily-related components, which are fed to the remainder current which is equal to the least significant bit. This current is shunted to ground, and the maximum output current is 255/256 of the reference amplifier current, or 1.992mA for a 2.0mA reference amplifier current if the NPN current source pair is perfectly matched.

Block Diagram:

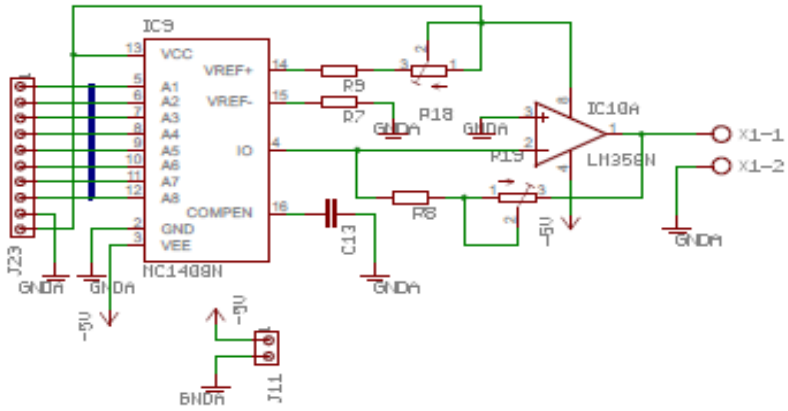


NPN Current Source Pair

DAC 1408N Pin Configurations:



Schematic Diagram:



Features:

- Fast settling time — 70ns (typ)
- Relative accuracy ±0.19% (max error)
- Non-inverting digital inputs are TTL and CMOS compatible
- High-speed multiplying rate 4.0mA/µs (input slew)
- Output voltage swing +0.5V to –5.0V
- Standard supply voltages +5.0V and –5.0V to –15V
- Military qualifications pending

Applications:

- Tracking A-to-D converters
- Waveform synthesis
- Sample-and-Hold
- Peak detector
- Programmable gain and attenuation
- Audio digitizing and decoding
- Programmable power supplies

Algorithm

1. Start
2. Store 0 in accumulator immediately.
3. Increment content of accumulator by one
4. Send content of accumulator to P2.
5. Check content of accumulator with FFH if true then go to step 4 else next step.
6. decrement content of accumulator by one
7. Send content of accumulator to P2.
8. Check content of accumulator with 00H if true then go to step 7 else next step.
9. Jump to step 2.

Program:

```
        org 00h
        mov a, #00h
up:     inc a
        mov p2,a
        cjne a,#0ffh,up
down:   dec a
        mov p2,a
        cjne a,#00h,down
        sjmp up
delay:
        mov r3,#0ffh
h2:     mov r4,#0ffh
h1:     djnz  r4,h1
        djnz r3,h2
        ret
        end
```

Result:

Triangular wave is generated using DAC interface with 8051 micro controller

Exercise:

1. DAC is based on ……. Circuitry.
2. Define resolution of DAC.
3. Define accuracy of DAC.

Lab Incharge **HOD**