**Single Chip Microcomputer and Embedded System**

**Experiment Reports**

**Major:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_­\_\_\_\_\_\_\_\_\_\_\_\_**

**Name: \_­­­­­­­­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Student ID: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Experiment II: 8051 Programming Language**

# Contents

Learn the programming techniques of the three structures of a program, and be able to use the basic structure to implement common programs such as code conversion and table lookup.

# Basic principle

Assembly Language is a pseudo-English representation of the Machine Language. The 8051 Microcontroller Assembly Language is a combination of English like words called Mnemonics and Hexadecimal codes.

It is also a low level language and requires extensive understanding of the architecture of the Microcontroller.

Although High-level languages are easy to work with, the following reasons point out the advantage of Assembly Language: The Programs written in Assembly gets executed faster and they occupy less memory. With the help of Assembly Language, you can directly exploit all the features of a Microcontroller. Using Assembly Language, you can have direct and accurate control of all the Microcontroller’s resources like I/O Ports, RAM, SFRs, etc. Compared to High-level Languages, Assembly Language has less rules and restrictions.

# Experimental steps

## Copy data from R0 of Bank0 to R0 of Bank3

Copy data from R0 of Bank0 to R0 of Bank3. Assembly programming code is as follows:

;--------------------------------------------------------------------------

ORG 00H

MOV R0, #33H

MOV A, R0

SETB PSW.3

SETB PSW.4

MOV R0, A

END

;--------------------------------------------------------------------------

## Toggle the LEDs ON and OFF that are connected to PORT1 of the 8051 MCU.

Use Proteus to create LEDs ON and OFF project. In the project, the LEDs are connected to P1 of AT89C51. In Keil software, create the corresponding project and generate HEX firmware. Simulate and observe the experimental results.

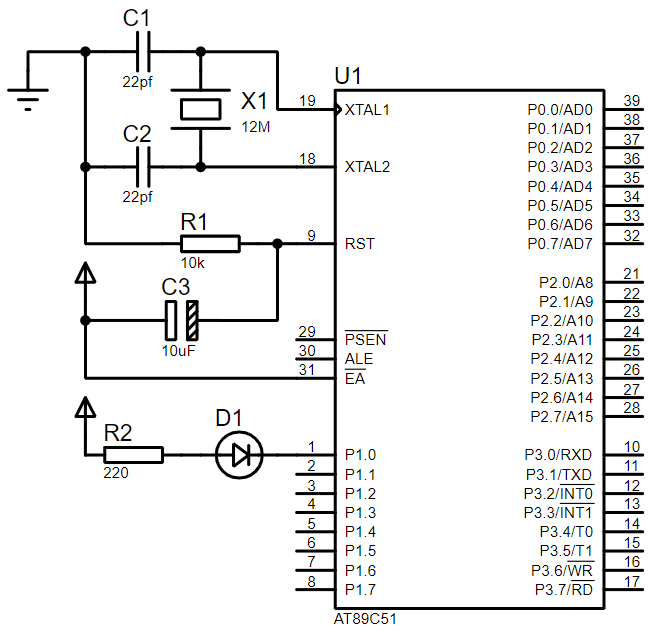


Fig. 1 Experimental schematic diagram

**3.2.1 Assembly Code**

;--------------------------------------------------------------------------

ORG 00H ; Assembly Starts from 0000H.

; Main Program

START: MOV P1, #0XFF ; Move 11111111 to PORT1.

CALL WAIT ; Call WAIT

MOV A, P1 ; Move P1 value to ACC

CPL A ; Complement ACC

MOV P1, A ; Move ACC value to P1

CALL WAIT ; Call WAIT

SJMP START ; Jump to START

WAIT: MOV R2, #10 ; Load Register R2 with 10 (0x0A)

WAIT1: MOV R3, #200 ; Load Register R3 with 10 (0xC8)

WAIT2: MOV R4, #200 ; Load Register R4 with 10 (0xC8)

DJNZ R4, $ ; Decrement R4 till it is 0. Stay there if not 0.

DJNZ R3, WAIT2 ; Decrement R3 till it is 0. Jump to WAIT2 if not 0.

DJNZ R2, WAIT1 ; Decrement R2 till it is 0. Jump to WAIT1 if not 0.

RET ; Retu rn to Main Program

END ; End Assembly

;--------------------------------------------------------------------------

**3.2.2 C code**

//-------------------------------------------------------------------------#include <reg52.h>

#define uchar unsigned char

#define uint unsigned int

sbit LED = P1^0;

void DelayMS(uint x)

{

uchar i;

while(x--)

{

for(i=120;i>0;i--);

}

}

void main()

{

while(1)

{

LED = ~LED;

DelayMS(150);

}

}

//-------------------------------------------------------------------------

# Experiment Results and Conclusion