LABORATORY MANUAL

ON

MICROCONTROLLER LAB

LAB INCHARGE: Dr Mukthi S.L

STAFF INCHARGE: Veena H.S — Associate professor

Dr Mukthi S.L — Assistant professor

Janardhan D — Assistant professor

SUBJECT CODE: 18ECL47

SEMESTER /YEAR: IV/FEB-2020

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

BANGALORE INSTITUTE OF TECHNOLOGY

BENGALURU

(Approved by AICTE, New Delhi & affiliated to VTU, Belagavi)

BANGALORE INSTITUTE OF TECHNOLOGY
BENGALURU

VISION

To Establish and Develop the Institute as a center of higher learning, ever abreast with expanding horizon of knowledge in the field of Engineering and Technology, with Entrepreneurial thinking, Leadership Excellence for life-long success and solve societal problem.

MISSION

- Provide high quality education in the Engineering disciplines from the undergraduate through doctoral levels with creative academic and professional programs.
- Develop the Institute as a leader in Science, Engineering, Technology and management, Research and apply knowledge for the benefit of society.
- Establish mutual beneficial partnerships with industry, alumni, local, state and central governments by public service assistance and collaborative research.
- Inculcate personality development through sports, cultural and extracurricular activities and engage in the social, economic and professional challenges.

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

VISION

Imparting **Quality Education** to achieve Academic Excellence in Electronics and Communication Engineering for Global Competent Engineers.

MISSION

- Create **state of art infrastructure** for quality education.
- Nurture **innovative concepts** and problem **solving skills**.
- Delivering **Professional Engineers** to meet the **societal needs.**

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO1	Prepare graduates to be professionals, practicing engineers and entrepreneurs in the field of Electronics and Communication.
PEO2	To acquire sufficient knowledge base for innovative techniques in design and development of tools and systems.
PEO3	Capable of competing globally in multidisciplinary field.
PEO4	Achieve personal and professional success with awareness and commitment to ethical and social responsibilities as an individual as well as a team.
PEO5	Graduates will maintain and improve technical competence. through continuous Learning process.

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO1	The graduates will be able to apply the principles of Electronics and Communication in core areas
PSO2	An ability to use latest hardware and software tools in Electronics and Communication engineering.
PSO3	Preparing Graduates to satisfy industrial needs and pursue higher studies with social-awareness and universal moral values.

PROGRAM OUTCOMES (POs)

POs	Description
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.
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 an modeling to complex engineering activities with an understanding of the limitations.
PO6	**The engineer and society:** Apply reasoning informed by the contextual knowledge to assess 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 team work:** 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.

SYLLABUS

B. E. (EC / TC)
Choice Based Credit System (CBCS) and Outcome Based Education (OBE)

SEMESTER – IV

MICROCONTROLLER LABORATORY

Laboratory Code	**18ECL47**	**CIE Marks**	**40**
Number of Lecture Hours/Week	**-+02Hr Tutorial (Instructions) + 02 Hours Laboratory**	**SEE Marks**	**60**
RBT Levels	**L1, L2, L3**	**Exam Hours**	**03**

CREDITS – 02

Course Learning Objectives: This laboratory course enables students to

- Understand the basics of microcontroller and its applications.
- Have in-depth knowledge of 8051 assembly language programming.
- Understand controlling the devices using C programming.
- The concepts of I/O interfacing for developing real time embedded systems.

Laboratory Experiments

1. Data Transfer: Block Move, Exchange, Sorting, Finding largest element in an array.
2. Arithmetic Instructions - Addition/subtraction, multiplication and division, square, Cube – (16 bits Arithmetic operations – bit addressable).
3. Counters.
4. Boolean & Logical Instructions (Bit manipulations).
5. Conditional CALL & RETURN.
6. Code conversion: BCD – ASCII; ASCII – Decimal; Decimal - ASCII; HEX - Decimal and Decimal - HEX.
7. Programs to generate delay, Programs using serial port and on-Chip timer/counter.

II. INTERFACING

1.Interface a simple toggle switch to 8051 and write an ALP to generate an interrupt which switches onan LED (i) continuously as long as switch is on and (ii) only once for a small time when the switch is turned on.
2.Write a C program to (i) transmit and (ii) to receive a set of characters serially by interfacing 8051 to a terminal.
3.Write ALPs to generate waveforms using ADC interface.
4.Write ALP to interface an LCD display and to display a message on it.
5.Write ALP to interface a Stepper Motor to 8051 to rotate the motor.
6.Write ALP to interface ADC-0804 and convert an analog input connected to it.

Course Outcomes: On the completion of this laboratory course, the students will be able to:

- Write Assembly language programs in 8051 for solving simple problems that manipulate input data using different instructions of 8051.
- Interface different input and output devices to 8051 and control them using Assembly language programs.
- Interface the serial devices to 8051 and do the serial transfer using C programming.

Conduct of Practical Examination:

- All laboratory experiments are to be included for practical examination.
- Students are allowed to pick one experiment from the lot.
- Strictly follow the instructions as printed on the cover page of answer script for breakup of marks.
- Change of experiment is allowed only once and 15% Marks allotted to the procedure part to be made zero.

COURSE OUTCOMES (COs)

215	18ECL47	MICROCONTROLLER LAB

Course Outcomes

Microcontroller Lab	18ECL47	C215.1	Write Assembly language programs in 8051 for solving simple problems that manipulate input data using different instructions of 8051.
		C215.2	Interface different input and output devices to 8051 and control them using Assembly language programs.
		C215.3	Write Assembly language programs in 8051 to generate timings and waveforms using 8051 Timer and to generate external interrupt using switch.
		C215.4	Interface the serial devices to 8051 and do the serial transfer using C programming.

CO-PO MAPPING

			PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
Microcontroller lab	18ECL47	C215.1	3												1	1	
		C215.2	2		1								1		1	1	
		C215.3	2		2								1		1	1	
		C215.4	2		2								1		1	1	
AVERAGE			2.25		1.66								1		1	1	

microcontroller lab	18ECL47	Justifications		Coverage of PO's	Coverage of PSO's	Taxonomy Levels	Blooms Taxonomy keywords
		C215.1	Apply the knowledge of Assembly language constructs and develop Assembly language programs using 8051.	P1-H	PSO1-L PSO2-L	L1,L2	Learn Apply develop
		C215.2	Understand the operations of I/O device /Interface the circuits and analyze using keil µv3 simulator	P1-M P3-L P11-L	PSO1-L PSO2-L	L2	Understand Develop analyze
		C215.3	Understand the interrupt system of 8051 and program to generate delay using Timer Interrupts and external hardware interrupt.	P1-M P3-M P11-L	PSO1-L PSO2-L	L2	Understand analyze
		C215.2	Develop C program to transmit and receive Data serially using 8051 serial port and terminal.	P1-M P3-M P11-L	PSO1-L PSO2-L	L2 L3	Develop

Course Handling Faculty **Course Co-ordinator** **Module Co-ordinator** **Program Co-ordinator**

DOs and DONTs

Students should follow the below DO's & DON'Ts:

- Follow the schedule time, late comers will not be permitted.
- Sign the Log book available in the Lab.
- Leave the Footwears in the specified stand before entering Lab.
- Keep belongings in the specified place.
- Students are expected to come prepared for experiments &VIVA.
- Show the completed Observation Book and submit Record to the Teacher before the Lab session begins.
- Cycle of experiments should be followed.
- Follow all the safety measures as suggested by the Teacher/Lab Instructor.
- Observe the instructions given by the Teacher/ Lab Instructor and strictly follow accordingly.
- Report to the Teacher/Lab Instructor immediately in case of any software/hardware/ Electrical failure during working. Never try to fix it manually/individually.
- Computers and any other experimental Kits should be switched OFF and chairs should be repositioned before leaving the Lab.
- Get the observations verified/signed by the teacher before leaving the Lab.
- Maintain Discipline & tidiness inside the Lab. Attend the Lab in formal attire.
- Usage of Mobile Phones, Pen Drive and Electronics Gadgets are restricted during the Lab session.

CONTENT

	III CYCLE	
4a.	Write an ALP to perform Boolean expression y=ab+$\overline{c}$ $\overline{d}$.	
4b.	Write an ALP to convert 8 bit Packed BCD number to ASCII.	
4c.	Write an ALP to convert ASCII to Decimal number.	
4d.	Write an ALP to convert Decimal number to ASCII	
4e.	Write an ALP to convert 8 bit Hexadecimal number to Decimal number.	
4f.	Write an ALP to convert 8 bit Decimal number (2 digit) to Hexadecimal number.	
	IV CYCLE	
5a.	An example program to demonstrate CALL and RETURN instruction.	
5b.	Write an ALP to generate delay of 0.5ms using software instructions. (generate square wave).Check the output on logic analyzer window	
5c.	(a).Write an ALP to generate delay of 1ms(0.25ms) using on chipTimer1. (b). Write an ALP to generate delay of 0.25ms using on chipTimer1 mode 2. (Generate square wave of 2KHz). Check the output on logic analyzer window.	
5d.	Write an ALP to transfer data serially at 4800 baud rate continuously. Check the output on serial window1.	
	V CYCLE	
6a.	Interface a simple toggle switch to 8051 and write an ALP to generate an interrupt which switches on an LED (i) continuously as long as switch is on and (ii) only once for a small time when the switch is turned on.	
6b.	Write a C program to (i) transmit and (ii) to receive a set of characters serially by interfacing 8051 to a terminal.	
6c.	Write ALPs to generate waveforms using DAC interface.	
6d.	Write ALP to interface an LCD display and to display a message on it.	
6e.	Write ALP to interface a Stepper Motor to 8051 to rotate the motor.	

Introduction

Atmel AT89C51ED2 - micro controller that has 64Kbytes of on-chip program memory and operates at 11.0592 MHz. It is a version of 8051 with enhanced features

PROCESSOR FEATURES

ON-CHIP MEMORY: CODE MEMORY: 64K Bytes of flash.

DATA MEMORY: 256 Bytes of RAM, 1792 Bytes of XRAM, 2K Bytes of EEPROM.

ON-CHIP PERIPHERALS

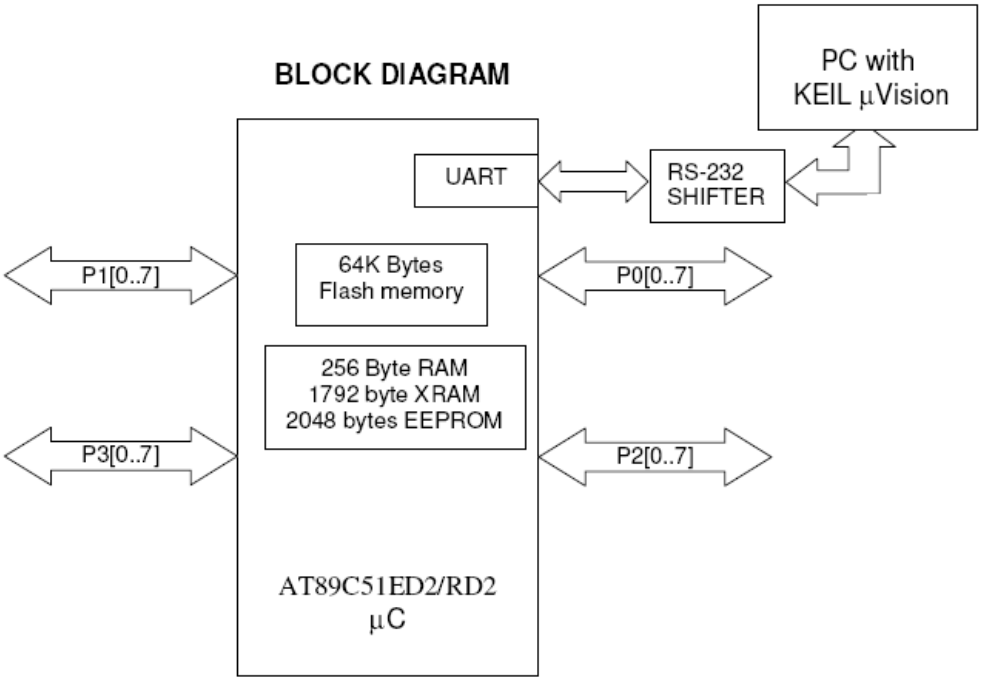
- Three 16-bit Timers/Counters, Watch Dog Timer, Programmable Counter Array (PCA) on Port1 i.e. PWM and Capture & Compare, SPI (Serial Peripheral Interface) on Port1, Full duplex enhanced UART.

INTERRUPTS

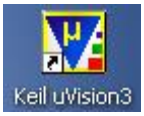
Nine sources of interrupt (both external and internal).
Two External interrupts INT0 and INT1 are provided with push button switches; these can also be used as general-purpose switches.
I/O (Port) Lines Four 10-pin connectors for all the 32 I/O lines.

P0, P1 and P2 Port lines are available on a 26-pin connector,

16X2 LCD & SERIAL I/O are also available.

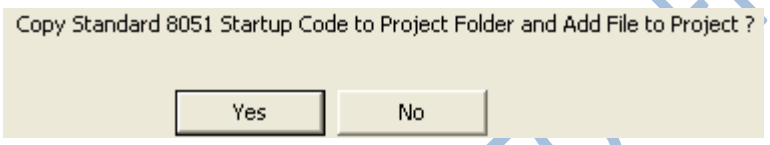


Creating and compiling a µ□Vision3 project

1. Double Click on the □□Vision3 icon on the desktop. 

2. Close any previous projects that were opened using – Project->Close.

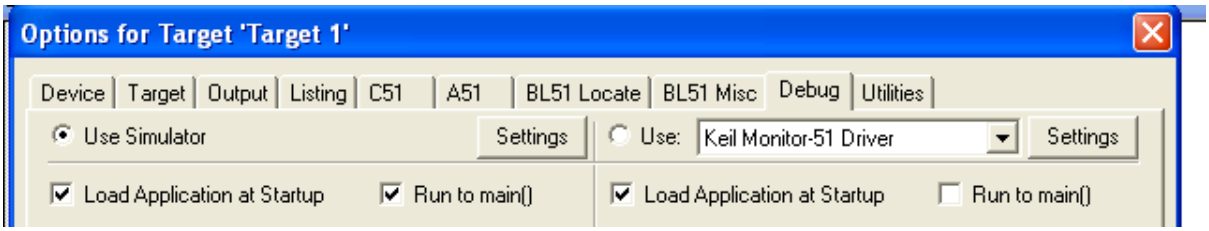
3. Start **Project – New Project,** and select the CPU from the device database (Database-Atmel- AT89C51ED2). (Select AT89C51ED2 or AT89C51RD2 as per the board).On clicking 'OK', the following option is displayed. Choose Yes.



4. Create a source file (using File->New), type in the assembly or C program and save this (filename.asm/ filename.c) and add this source file to the project using either one of the following two methods. (i) Project-Components,Environmentand Books->addfiles-> browse to the required file -> OK "OR"

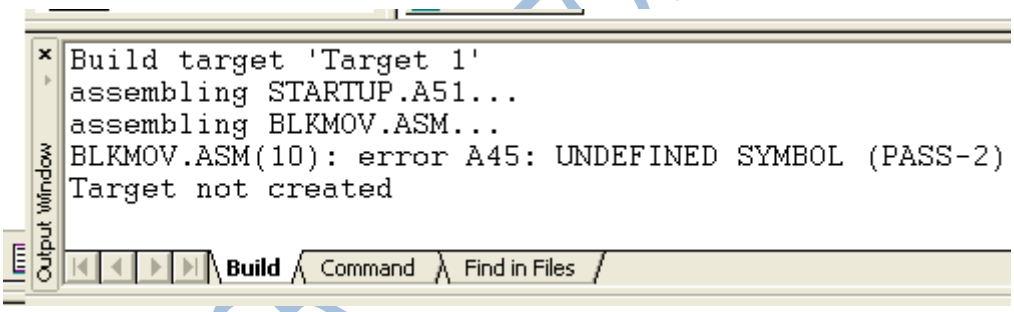
(ii) right click on the Source Group in the Project Window and the **Add Files to Group** option. 

5. Set the Target options using -> **Project** – **Options for Target** opens the □□Vision2 **Options for Target – Target** configuration dialog. Set the Xtal frequency as 11.0592 Mhz, and also the **Options for Target – Debug – use either Simulator / Keil Monitor- 51 driver.**



If **Keil Monitor- 51 driver is used click on Settings** -> COM Port settings select the COM Port to which the board is connected and select the baud rate as 19200 or 9600 (recommended). Enable **Serial Interrupt** option if the user application is not using on-chip UART, to stop program execution.

6. Build the project; using Project -> Build Project. □□Vision translates all the user application and links. Any errors in the code are indicated by – "Target not created" in the Build window, along with the error line. Debug the errors. After an error free build, goto Debug mode



7. Now user can enter into **Debug** mode with **Debug- Start / Stop Debug session** dialog. Or by clicking in the  icon.

8.The program is run using the **Debug-Run** command & halted using **Debug-Stop Running.** Also the  (reset, run, halt) icons can be used. Additional icons are  (step, step over, step into, run till cursor).

9. If it is an interface program the outputs can be seen on the LCD, CRO, motor, led status, etc. If it is a part A program, the appropriate memory window is opened using View -> memory window (for data RAM & XRAM locations), Watch window (for timer program), serial window, etc.

Note: To access data RAM area type address as D:0020h.

Similarly to access the DPTR region (XRAM-present on chip in AT89C51ED2) say 8000h location type in X:08000H.

MOV dest,src---------FORMAT in 8051

I CYCLE

1a. Write an assembly language program to transfer n =16 bytes of data from location 50h to location 60h (without overlap).

```
        Org 0000h           ;PC loaded with 0000h the starting address of code memory
        Sjmp 30h
        Org 30h
        mov r0,#50h         ;r0 pointing  to source block at 50h (internal RAM)
        mov r1,#60h         ;r1 pointing to destination block at 60h
        mov r2,#10h         ;r2 loaded with no. of elements in the array
back:
        mov a,@r0           ;data transfer from src block to accumulator and
        mov @r1,a           ;then to dst block
        inc r0              ;increment pointers.
        inc r1
        djnz  r2,back       ;decrement r2,if not equal to 0,continue with data
                            ;transfer process.
here:   sjmp  here
        end
```

1a.Algorithm

1. Initialize registers to the source & destination addresses and the number of bytes to be transferred (count).
2. Get data from source location into accumulator and transfer to the destination location.
3. Decrement the count register and repeat step 2 till count is zero.

Note: For data transfer with overlap start transferring data from the last location of source array to the last location of the destination array.

RESULT:

Before Execution:

Address: D:50h

```
D:0x50: 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 00 00 00 00
D:0x64: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 00 00 00 00
D:0x8C: 00 00 08 00 FF 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00
D:0xA0: FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 00
D:0xB4: 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 14 00 00 00 FF
D:0xC8: 00 00 00 00 00 00 00 00 00 F0 00 00 00 00 00 00 00 00 00 00
```

After Execution:

Address: D:60h

```
D:0x60: 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 00 00 00 00
D:0x74: 00 00 00 00 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10
D:0x88: 00 00 00 00 00 00 08 00 FF 00 00 00 00 00 00 FF 00 00 00 00
D:0x9C: 00 00 00 00 FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0xB0: FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 14
D:0xC4: 00 00 00 FF 00 00 00 00 00 00 00 00 00 F0 00 00 00 00 00 00
D:0xD8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 00
```

1b. Write an assembly language program to transfer n =16 bytes of data from external memory location 8050h to location 8060h (without overlap).

```
        org 0000h           ;PC loaded with 0000h the starting address of code
                              memory location
        sjmp 30h
        org 30h
        mov dptr,#8050h     ;DPTR pointed to external RAM location 8050h(source)
        mov r0,# 80h        ;save  destination location 8060h at r0 and r1
        mov r1,#60h
        mov r7,#10h         ;load the no. of  data to be transferred to r7=05h(bytes)
back:   movx a,@dptr        ; transfer the data from src location 8050h to accumulator
        inc dptr            ; increment src pointer.
        mov r2,dph          ; store src  addr at r2 & r3
        mov r3,dpl

        mov dph,r0          ; get the dest addr to dptr
        mov dpl,r1
        movx @dptr          ; transfer from 'a' to dest addr
        inc dptr
        mov r0,dph          ; store dest address at r0 & r1
        mov r1,dpl

        mov dph,r2          ; get back src addr to dptr
        mov dpl,r3
        djnz  r7,back       ; decrement counter if r7≠0,continue the data transfer
here:   sjmp here           ; r7=0, all the data are transferred .
        end
```

1b.Algorithm

1. Initialize registers to the source & destination address (external memory) and the number of bytes to be transferred(count).
2. Get data from source location into accumulator and transfer to the destination location.
3. Decrement the counter register and repeat step 2 till count is zero.

RESULT:

Before Execution:

Address: x:8050h

```
X:0x008050: 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 00 00 00
X:0x008063: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
X:0x008076: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
X:0x008089: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
X:0x00809C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
X:0x0080AF: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
X:0x0080C2: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

After Execution:

Address: x:8060h

```
X:0x008060: 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 00 00 00 00
X:0x008074: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
X:0x008088: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
X:0x00809C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
X:0x0080B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
X:0x0080C4: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
X:0x0080D8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

1c. Write an assembly language program to exchange 16 bytes of data from 50h to 60h.

```
        org 0000h       ; PC loaded with 0000h the starting address of code memory
                          location
        sjmp 30h
        org 30h
        mov r0,#50h     ;r0 pointing  to one block of data at 50h (internal RAM)
        mov r1,#60h       ;r1 pointing to another block of data at 60h
        mov r7,#10h       ; initialize r7=10h, as counter the no. of data to be exchanged
back:   mov a,@r0         ; exchange the data between one locn  with another locn.
        xch a,@r1
        mov @r0,a
        inc r0
        inc r1
        djnz r7,back      ;decrement counter, if r7≠0,continue the exchange of data
here:   sjmp here         ; r7=0 and data exchange is completed
        end
```

1c.Algorithm

1. Initialize registers to the source & destination addresses(internal memory) and the number of bytes to be exchanged(count).
2. Exchange data between the source location and the destination location.
3. Decrement the counter register and repeat step 2 till count is zero.

RESULT:

Before Execution:

Address: d:50h

```
D:0x50: 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF AD 01 02 03 04
D:0x64: 05 06 07 08 09 10 1A 1B 1C 1D 1E 1F 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 00 00 00 00
D:0x8C: 00 00 08 00 FF 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00
D:0xA0: FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 00
D:0xB4: 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 14 00 00 00 FF
D:0xC8: 00 00 00 00 00 00 00 00 00 F0 00 00 00 00 00 00 00 00 00 00
```

After Execution:

Address: d:50h

```
D:0x50: 01 02 03 04 05 06 07 08 09 10 1A 1B 1C 1D 1E 1F 11 22 33 44
D:0x64: 55 66 77 88 99 AA BB CC DD EE FF AD 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 00 00 00 00
D:0x8C: 00 00 08 00 FF 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00
D:0xA0: FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 00
D:0xB4: 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 14 00 00 00 FF
D:0xC8: 00 00 00 00 00 00 00 00 01 F0 00 00 00 00 00 00 00 00 00 00
```

1d. Write an ALP to determine the largest number in an array of N=10numbers and find its position (N can be variable).

```
                org 0000h
                sjmp 30h
                org 30h
                mov r0,#50h          ; r0 is pointing to array of data  stored at 50h
                mov r7,#0Ah          ;initialise a counter r7=0Ah
                mov a, #00h          ;mov a,#0ffh for smallest
again:          mov b,@r0
                cjne a,b,label       ;compare -data at A register  > data at B register
                sjmp next
label:          jnc next             ;for smallest use jc instruction
                mov a,b              ;save the largest no.at  A register
                mov r4,00h           ;save its position at reg r4(00h-direct address of r0)
next:           inc r0
                djnz r7,again        ;check the largest among  all the elements in the array
                mov 60h,a            ; save the largest number at 60h data memory location
                mov 61h,04h          ;save its position at 61h data memory location from
                                     ; r4-Direct address
                sjmp $
                end
```

1d.Algorithm

1. Initialize register r0 =50h(internal memory)where array of elements are stored and initialize the counter for number of elements in the array.

2. Compare the numbers at consecutive memory location check the larger number and exchange compare all numbers
3. Decrement counter, and repeat step 2 until counter=0,also Save the largest number at data memory location 60h and its position at 61h

NOTE: For smallest number use instruction **JC next** instead of **JNC next** and repeat the same way.

Note: 60h – Largest number
61h – Position of the largest number

RESULT:

Before Execution:

Address: d:50h

```
D:0x50: 11 54 33 85 FF 12 67 89 34 60 00 00 00 00 00 00 00 00 00 00
D:0x64: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 00 00 00 00
D:0x8C: 00 00 08 00 FF 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00
D:0xA0: FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 00
D:0xB4: 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 14 00 00 00 FF
D:0xC8: 00 00 00 00 00 00 00 00 00 F0 00 00 00 00 00 00 00 00 00 00
```

After Execution:

Address: d:50h

```
D:0x50: 11 54 33 85 FF 12 67 89 34 60 00 00 00 00 00 00 FF 54 00 00
D:0x64: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 00 00 00 00
D:0x8C: 00 00 08 00 FF 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00
D:0xA0: FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 00
D:0xB4: 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 14 00 00 00 FF
D:0xC8: 00 00 00 00 00 00 00 00 00 F0 00 00 00 00 00 00 00 00 00 00
```

1e. Write an assembly language program to sort an array of 10 numbers stored at data memory location 50h in increasing / decreasing order (using bubble sort).

```
                org 0000h
                sjmp start
                org 30h
start:          mov r0,#50h          ;r0 is pointing to unsorted array of data  stored at 50h
                mov r2,#09h          ;load r2=09h the pass counter(count=total num-1)
outer:          mov r3,#09h          ;load r3=09h the comparison counter(count=total num-1)

inner:          mov b,@r0
                inc r0
                mov a,@r0            ;compare two data's, if  contents of A register is greater
                cjne a,b,next        ; than B register exchange ,else check with the next data

next:           jc  nochange        ;JNC for Ascending order
                mov @r0,b
                dec r0
                mov @r0,a
                inc r0
nochange:       djnz  r3,inner       ; decrement comparison counter until r3=0
                mov r0,#50h
                djnz r2,outer        ; decrement  pass counter
                sjmp $
                end
```

1e.Algorithm

1. Initialize register r0 =50h(internal memory)where unsorted array is stored and initialize the pass Counter and comparison counter

2. Compare the numbers at consecutive memory location check the larger number and exchange. Compare all numbers in every pass.
3. Decrement comparison counter, and repeat step 2 until counter=0,also decrement pass Counter after every pass

4. If pass counter≠0 ,repeat step 2 &3 until pass counter=0
5. Save the sorted array from data memory location from 50h onwards in data memory location

Note: Sorting for descending order use instruction **JC** exchange instead of **JNC** exchange and repeat the same as above.

RESULT:

Before Execution:

Address: d:50h

```
D:0x50: 99 55 66 33 11 44 77 22 88 31 00 00 00 00 00 00 00 00 00 00
D:0x64: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 00 00 00 00
D:0x8C: 00 00 08 00 FF 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00
D:0xA0: FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 00
D:0xB4: 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 14 00 00 00 FF
D:0xC8: 00 00 00 00 00 00 00 00 00 F0 00 00 00 00 00 00 00 00 00 00
```

After Execution:

Address: d:50h

```
D:0x50: 99 88 77 66 55 44 33 31 22 11 00 00 00 00 00 00 00 00 00 00
D:0x64: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 00 00 00 00
D:0x8C: 00 00 08 00 FF 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00
D:0xA0: FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 00
D:0xB4: 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 14 00 00 00 FF
D:0xC8: 00 00 00 00 00 00 00 00 80 F0 00 00 00 00 00 00 00 00 00 00
```

II CYCLE

2a. Write an ALP to perform addition of N=2, 16 bit numbers stored from data memory location 50h (higher first). Store the result from data memory location 60h.

```
        org 0000h
        sjmp 30h
        org 30h
        mov r0,#51h          ;r0 pointed to internal RAM loc 51h (LS byte of data1)
        mov r1,#53h          ;r1 pointed to internal RAM loc 53h (LSbyte of data2)
        clr c
        mov a,@r0            ;perform LS byte addition save the result at 62h location
        add a,@r1
        dec r0
        dec r1
        mov 62h,a
        mov a,@r0            ;perform MS byte addition save the result at 61h loc
        addc a,@r1
        mov 61h,a
        jnb 0D7h, skip       ; JNB 0D7h,target is bit addressable instruction equivalent
                               to JNC instruction
        mov 60h,#01h         ; if carry is there after addition save carry at 60h loc
        sjmp here
skip:   mov 60h,#00h
here:   sjmp here
        end
```

Note:

Ex: **(A512+6C89)**
Data: 50h 51h
Data: 52h 53h
Result: 61h 62h - sum
60h - carry

RESULT:

Before Execution:

Address: D:50h

```
D:0x50: A5 12 6C 89 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x64: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 00 00 00 00
D:0x8C: 00 00 08 00 FF 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00
D:0xA0: FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 00
D:0xB4: 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 14 00 00 00 FF
D:0xC8: 00 00 00 00 00 00 00 00 00 F0 00 00 00 00 00 00 00 00 00 00
```

After Execution:

Address: D:50h

```
D:0x50: A5 12 6C 89 00 00 00 00 00 00 00 00 00 00 00 00 01 11 9B 00
D:0x64: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 00 00 00 00
D:0x8C: 00 00 08 00 FF 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00
D:0xA0: FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 00
D:0xB4: 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 14 00 00 00 FF
D:0xC8: 00 00 00 00 00 00 00 00 C0 F0 00 00 00 00 00 00 00 00 00 00
```

2b. Write an ALP to perform subtraction of N=2, 16 bit numbers stored from data memory location 50h (higher byte first). Store the result from data memory location 60h.

```
        org 0000h
        sjmp 30h
        org 30h
        mov r0,#51h          ;r0 pointed to internal RAM loc 51h (LS byte of data1)
        mov r1,#53h          ;r1 pointed to internal RAM loc 53h (LS byte of data2)
        clr c
        mov a,@r0            ;perform LS byte subtraction save the result at 62h location
        subb a,@r1
        dec r0
        dec r1
        mov 62h,a
        mov a,@r0            ;perform MS byte subtraction save the result at 61h loc
        subb a,@r1
        mov 61h,a
        jnb 0d7h, skip       ;JNB 0D7h,target is bit addressable instruction equivalent
                             to JNC instruction

        mov 60h,#01h         ; if borrow is there after subtraction save borrow at 60h loc
        sjmp here
skip:   mov 60h,#00h
here:   sjmp here
        end
```

Note:

EX: (6C89 – 284A**)**
Data: 50h 51h
Data: 52h 53h
Result: 61h 62h - Difference
60h - Barrow

RESULT:

Before Execution:

Address: D:50h

```
D:0x50: 6C 89 28 4A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x64: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 00 00 00 00
D:0x8C: 00 00 08 00 FF 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00
D:0xA0: FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 00
D:0xB4: 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 14 00 00 00 FF
D:0xC8: 00 00 00 00 00 00 00 00 00 F0 00 00 00 00 00 00 00 00 00 00
```

After Execution:

Address: D:50h

```
D:0x50: 6C 89 28 4A 00 00 00 00 00 00 00 00 00 00 00 00 00 44 3F 00
D:0x64: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 00 00 00 00
D:0x8C: 00 00 08 00 FF 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00
D:0xA0: FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 00
D:0xB4: 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 14 00 00 00 FF
D:0xC8: 00 00 00 00 00 00 00 00 00 F0 00 00 00 00 00 00 00 00 00 00
```

2c.Write an ALP to perform Multiplication of 16 bit number at 50h(50h-MSB,51h-LSB) and 8 bit number at 52h. Save the result at 60h(MSB2),61h(MSB1),62h(LSB).

```
        org 0000h
        sjmp 30h
        org 30h
        mov a,51h       ; get LS byte of multiplicand to A register
        mov b,52h       ; get multiplier to B register
        mul ab          ; multiply
        mov 61h,b       ; save MS byte of the result at 61h
        mov 62h,a       ; save LSB byte of the result at 62h(LSB of result)
        mov a,50h       ; get MS byte of the multiplicand to A register
        mov b,52h       ; get multiplier to B register
        mul ab          ; multiply
        add a,61h       ; after multiplication add contents of A register to 61h
        mov 61h,a       ; save MSB1 of the result at 61h
        mov a,0f0h
        addc a,#00h
        mov 60h,a       ; save MSB2 of the result at 60h
        sjmp $
        end
```

Note:
Ex: (FFEE*DD)
Data: 50h 51h(multiplicand)
Data: 52h (multiplier)
Result: 60h 61h 62h

RESULT:

Before Execution:

Address: D:50h

```
D:0x50: FF EE DD 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x64: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 00 00 00 00
D:0x8C: 00 00 08 00 FF 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00
D:0xA0: FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 00
D:0xB4: 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 14 00 00 00 FF
D:0xC8: 00 00 00 00 00 00 00 00 00 F0 00 00 00 00 00 00 00 00 00 00
```

After Execution:

Address: D:50h

```
D:0x50: FF EE DD 00 00 00 00 00 00 00 00 00 00 00 00 00 DC F0 76 00
D:0x64: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 00 00 00 00
D:0x8C: 00 00 08 00 FF 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00
D:0xA0: FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 00
D:0xB4: 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 14 00 00 00 FF
D:0xC8: 00 00 00 00 00 00 00 00 01 F0 00 00 00 00 00 00 00 00 00 00
```

2d.Write an ALP to perform 16bit/8bit Division.(MSB of dividend-50h,LSB of dividend-51h,divisor-52h).

```
        org 0000h
        sjmp 30h
        org 30h
        mov r0,#51h          ; LSByte  of 16 bit dividend,
        mov r1,#52h          ; 8bit divisor is stored
        mov dptr,#0000h      ;initialize  DPTR reg.=0000h for quotient
        clr c
repeat: mov a,@r0            ; get lower byte of dividend
        subb a,@r1           ;subtract divisor from dividend
        mov @r0,a            ; save result at 51hand 50h
        dec r0
        mov a,@r0
        subb a,#00h
        mov @r0,a
        inc r0
        jc last              ;after every subtraction check if borrow flag is set
        inc dptr             ;after every sub increment dptr (quotient-no. of subtraction)
        sjmp repeat
last:   mov a,@r0            ; if borrow flag is set, add divisor to result to get  remainder
        add a,@r1
        mov 63h,a            ; save LSB of remainder at 63h
        dec r0
        mov a,@r0
        addc a,#00h          ;save MSB of remainder at 62h
        mov 62h,a
        mov 61h,dpl          ; save LSB of quotient at 61h
        mov 60h,dph          ; save MSB of quotient at 60h
        sjmp $
        end
```

Note:
EX: (29f0h*09)
Data: 50h 51h (dividend)
Data: 52h (divisor)
Result: 60h - MSB of quotient
61h - LSB of quotient
62h - MSB of remainder
63h - LSB of remainder

RESULT:

Before Execution:

Address: d:50h

```
D:0x50: 29 F0 09 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x64: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 00 00 00 00
D:0x8C: 00 00 08 00 FF 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00
D:0xA0: FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 00
D:0xB4: 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 14 00 00 00 FF
D:0xC8: 00 00 00 00 00 00 00 00 00 F0 00 00 00 00 00 00 00 00 00 00
```

After Execution:

Address: d:50h

```
D:0x50: FF FF 09 00 00 00 00 00 00 00 00 00 00 00 00 00 04 A8 00 08
D:0x64: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 FF 07 A8 04 00 01 01 10 00 00 00 00
D:0x8C: 00 00 08 00 FF 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00
D:0xA0: FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 00
D:0xB4: 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 14 00 00 00 FF
D:0xC8: 00 00 00 00 00 00 00 00 C0 F0 00 00 00 00 00 00 00 00 00 00
```

2e.Write an ALP to perform square and cube of a 8 bit number.

```
        org 000h
        sjmp 30h
        org 30h
        mov a,50h           ;get multiplicand to A register
        mov b,50h           ;get multiplier to A register
        mul ab              ; multiply (square)
        mov 61h,a           ;store lower byte  of square of a number in higher address
        mov 60h,b           ;store higher byte  of square of a number in lower address
        mov b,50h
        mul ab
        mov 64h,a           ;store lower byte of cube in higher address
        mov 63h,b
        mov b,50h
        mov a,60h
        mul ab
        add a,63h
        mov 63h,a
        mov a,b
        addc a,#00h
        mov 62h,a           ;store higher byte of cube in lower address
        sjmp $
        end
```

Note: 61h – LS byte of square
60h – MS byte of square
63h, 64h – Lower byte of cube
62h – Higher byte of cube

RESULT:

Before Execution:

Address: D:50h

```
D:0x50: FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x64: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 00 00 00 00
D:0x8C: 00 00 08 00 FF 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00
D:0xA0: FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 00
D:0xB4: 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 14 00 00 00 FF
D:0xC8: 00 00 00 00 00 00 00 00 00 F0 00 00 00 00 00 00 00 00 00 00
```

After Execution:

Address: D:50h

```
D:0x50: FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FE 01 FD 02
D:0x64: FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 00 00 00 00
D:0x8C: 00 00 08 00 FF 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00
D:0xA0: FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 00
D:0xB4: 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 14 00 00 00 FF
D:0xC8: 00 00 00 00 00 00 00 00 01 F0 00 00 00 00 00 00 00 00 00 00
```

3a. Write an ALP to output the binary up counting(00h to 0ffh) sequence on port1 and observe o/p on logic analyzer window.(observe the delay)

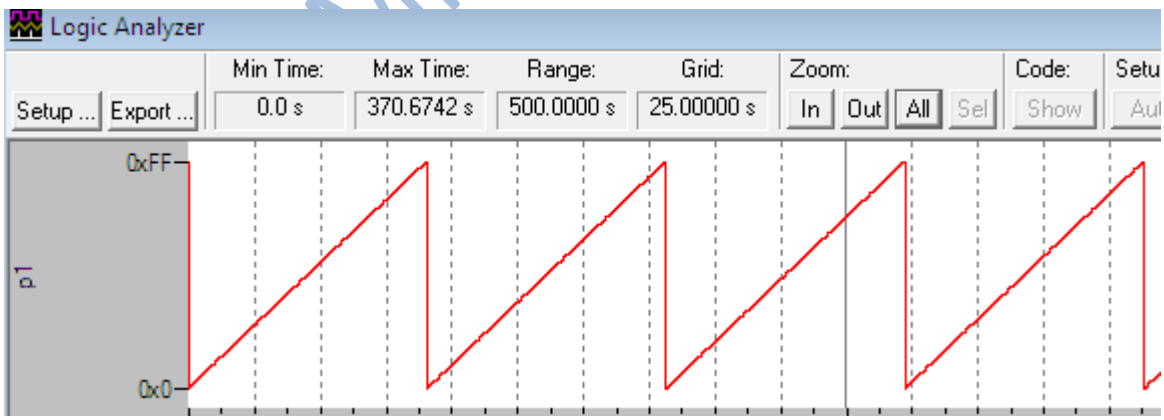
```
        org 0000h
        sjmp 30h
        org 30h
        mov p1,#00h          ;configure p1 as o/p port
        mov a,#00h
back:   mov p1,a             ;load p1 with initial count
        acall delay          ; give delay between count
        inc a                ;add 01 to increment the count
        sjmp  back
delay:  mov r4,#0ffh         ;delay between counts
repeat: mov r5,#0ffh
next:   mov r6,#04h
l1:     nop
        djnz r6,l1
        djnz r5,next
        djnz r4,repeat
        ret
        end
```

Time delay calculation:

Delay program	Machine cycle
delay: mov r4,#0ffh	1
repeat: mov r5,#0ffh	1
next: mov r6,#04h	1
l1: nop	1
djnz r6,l1	2
djnz r5,next	2
djnz r4,repeat	2
ret	2

Time delay: (255(255((2+1)4)+1+2)+1+2)+1+2+2) x 1.0852µs

RESULT:

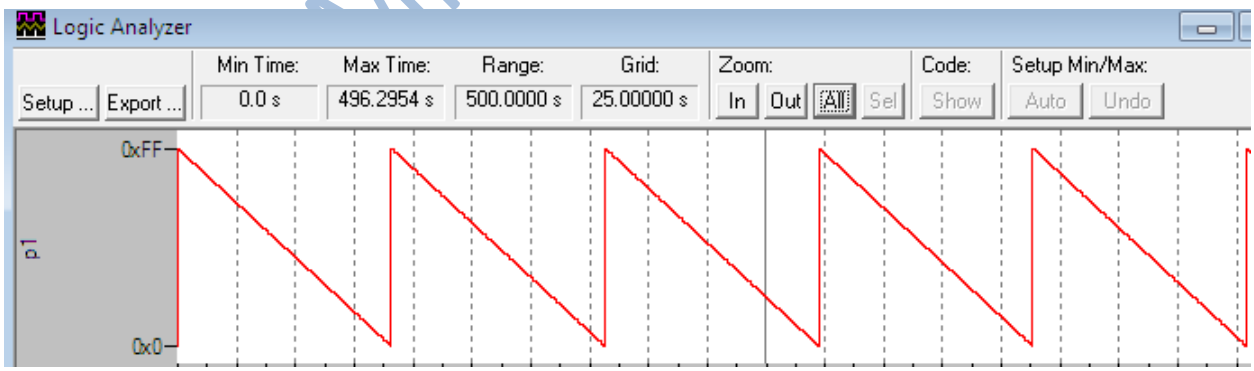


3b.Write an ALP to output the binary down(0ffh to 00h) counting sequence on port1 and observe o/p on logic analyzer window.(observe the delay)

```
        org 0000h
        sjmp 30h
        org 30h
        mov p1,#00h          ;configure p1 as o/p port
        mov a,#0ffh
back:   mov p1,a             ;load p1 with initial count
        acall delay          ; delay between count
        dec  a               ;subtract 01 to decrement the     count
        sjmp  back

delay:  mov r4,#0ffh         ;delay between counts
repeat:  mov r5,#0ffh
next:   mov r6,#04h
l1:     nop
        djnz r6,l1
        djnz r5,next
        djnz r4,repeat
        ret
        end
```

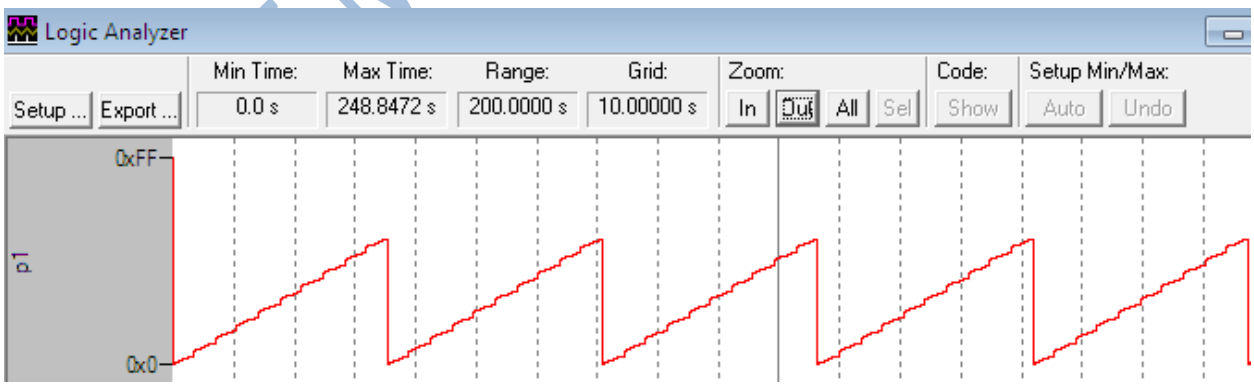
RESULT:



3c.Write an ALP to output the BCD up counting(00 to 99) sequence on port1 and Observe o/p on logic analyzer window.(observe the delay)

```
        org 0000h
        sjmp 30h
        org 30h
        mov p1,#00h          ;configure p1 as o/p port
        mov a,#00h
back:   mov p1,a             ;load p1 with initial count
        acall delay          ; delay
        add a,#01            ;add 01 to increment the count
        da a
        sjmp back
delay:  mov r4,#0ffh         ;delay between counts
repeat: mov r5,#0ffh
next:   mov r6,#04h
l1:     nop
        djnz r6,l1
        djnz r5,next
        djnz r4,repeat
        ret
        end
```

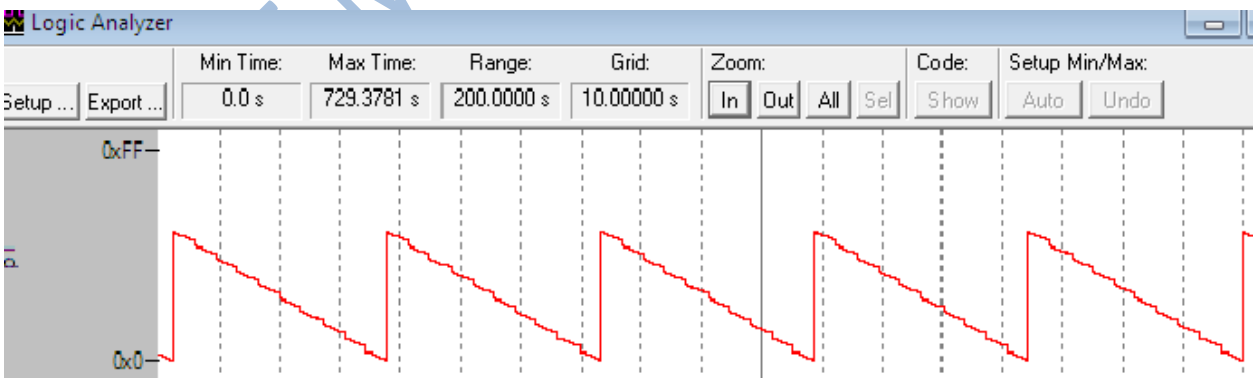
RESULT:



3d.Write an ALP to output the BCD down counting(99 to 00) sequence on port1 and observe o/p on logic analyzer window. (observe the delay)

```
        org 0000h
        sjmp 30h
        org 30h
start:  mov p1,#00h          ;configure p1 as o/p port
        mov a,#99h
back:   mov p1,a             ;load p1 with initial count
        acall delay          ; delay
        add a,#99h           ;add 01 to increment the count
        da a
        sjmp  back
delay:  mov r4,#0ffh         ;delay between counts
repeat:  mov r5,#0ffh
next:   mov r6,#04h
l1:     nop
        djnz r6,l1
        djnz r5,next
        djnz r4,repeat
        ret
        end
```


RESULT:



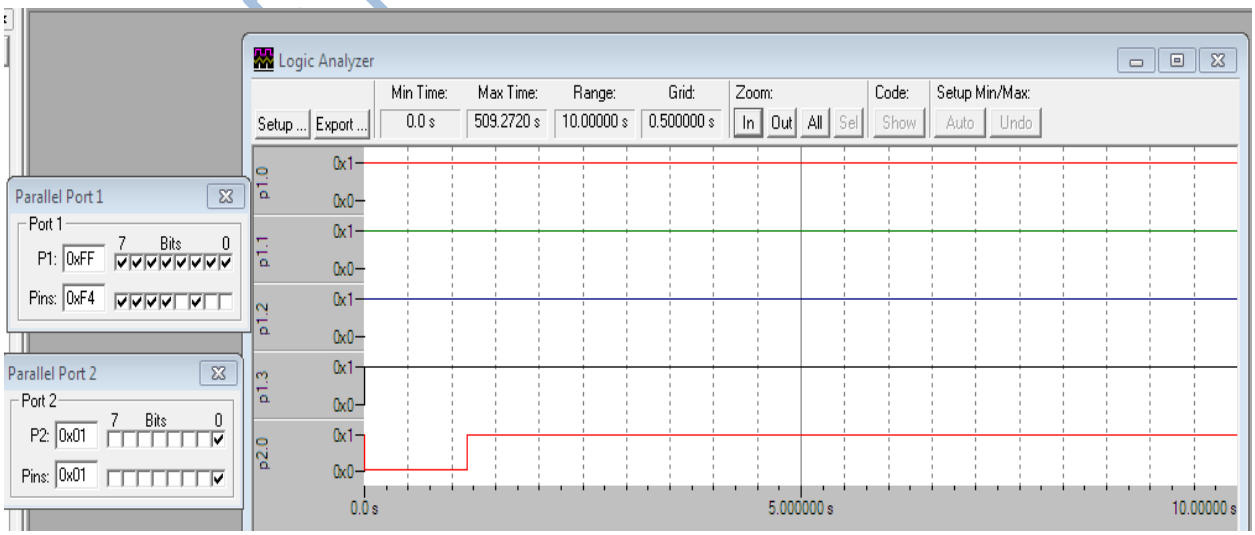
III CYCLE

4a.Write an ALP to perform Boolean expression Z=ab+$\bar{c}$ $\bar{d}$.

```
                org 0000h
                sjmp 30h
                org 30h
start:          mov p2,#00h          ;Initialize port P2 as O/P port and port P1 as I/P port
                mov p1,#0ffh         ;read port P1 as  d=P1.0,c=P1.1,b=P1.2,a=P1.3
repeat:         jnb p1.3 ,second     ;check if variable a=0,if 0 check c and d
                jnb p1.2,second      ;check if variable b=0 ,if 0 check c and d
                setb p2.0            ; if a=1,b=1 set o/p z=1
                sjmp repeat          ;if true read   port P1i.e next i/p
second:         jb p1.1,false        ;check if c=1
                jb p1.0,false        ;check if d=1
                setb p2.0            ;if c=0,d=0 set o/p z=1
                sjmp exit            ;repeat for next set of i/p
false:          clr p2.0             ;if a=0,b=0,c=1,d=1   clear o/p i.e z=0
exit:           sjmp repeat          ;read port for the next combination of input
                end
```

Note: P1- Input port
P2- Output port

RESULT:



4b.Write an ALP to convert 8 bit Packed BCD number to ASCII.

```
        org 000h
        sjmp 30h
        org 30h
        mov a,50h       ; load to accumulator the BCD number from data memory location
                          50h
        anl a,#0fh
        add a,#30h
        mov 61h,a       ;store the lower nibble ACSII value in higher address
        mov a,50h
        anl a,#0f0h
        swap a
        add a,#30h
        mov 60h,a       ;store the higher nibble ACSII value in lower address
here:   sjmp here
        end
```

RESULT:

Before Execution:

Address: d:50h

```
D:0x50: 82 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x64: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 00 00 00 00
D:0x8C: 00 00 08 00 FF 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00
D:0xA0: FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 00
D:0xB4: 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 14 00 00 00 FF
D:0xC8: 00 00 00 00 00 00 00 00 00 F0 00 00 00 00 00 00 00 00 00 00
```

After Execution:

Address: d:50h

```
D:0x50: 82 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 38 32 00 00
D:0x64: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 00 00 00 00
D:0x8C: 00 00 08 00 FF 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00
D:0xA0: FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 00
D:0xB4: 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 14 00 00 00 FF
D:0xC8: 00 00 00 00 00 00 00 00 01 F0 00 00 00 00 00 00 00 00 00 00
```

4c.Write an ALP to convert ASCII to Decimal number.

```
        org 000h
        sjmp 30h
        org  30h
        mov a,50h          ; load the ASCII number to accumulator from 50h
        cjne a,#40h,label  ; check the ASCII number is > 40
label:  jc exit
        clr c
        subb a,#37h        ; if > 40 subtract 37h to convert to hex value and save at 51h
        sjmp last
exit :  clr c
        subb a,#30h        ; else subtract 30h to convert to hex value and save at 51h
last:   mov 51h,a
        sjmp $
        end
```

RESULT:

Before Execution:

Address: d:50h

```
D:0x50: 43 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x64: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 00 00 00 00
D:0x8C: 00 00 08 00 FF 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00
D:0xA0: FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 00
D:0xB4: 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 14 00 00 00 FF
D:0xC8: 00 00 00 00 00 00 00 00 00 F0 00 00 00 00 00 00 00 00 00 00
```

After Execution:

Address: d:50h

```
D:0x50: 43 0C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x64: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 00 00 00 00
D:0x8C: 00 00 08 00 FF 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00
D:0xA0: FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 00
D:0xB4: 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 14 00 00 00 FF
D:0xC8: 00 00 00 00 00 00 00 00 40 F0 00 00 00 00 00 00 00 00 00 00
```

\

4d.Write an ALP to convert Decimal number to ASCII

```
org 000h
sjmp 30h
org 30h
mov a,50h          ; load accl the decimal number from 50h
anl a,#0fh
add a,#30h         ;mask upper byte
mov 52h,a          ; save ASCII value of LS byte of  number to 52h
mov a,50h
anl a,#0f0h
swap a
add a,#30h
mov 51h,a        ; save ASCII value of MS byte of  number to 52h
H:sjmp H
End
```

RESULT:

Before Execution:

Address: d:50h

```
D:0x50: 32 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x63: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x76: 00 00 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 00
D:0x89: 00 00 00 00 00 08 00 FF 00 00 00 00 00 00 FF 00 00 00 00
D:0x9C: 00 00 00 00 FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0xAF: 00 FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 00
D:0xC2: 00 14 00 00 00 FF 00 00 00 00 00 00 00 00 00 F0 00 00 00
```

After Execution:

Address: d:50h

```
D:0x50: 32 33 32 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x63: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x76: 00 00 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 00
D:0x89: 00 00 00 00 00 08 00 FF 00 00 00 00 00 00 FF 00 00 00 00
D:0x9C: 00 00 00 00 FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0xAF: 00 FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 00
D:0xC2: 00 14 00 00 00 FF 00 00 00 00 00 00 00 00 00 F0 00 00 00
```

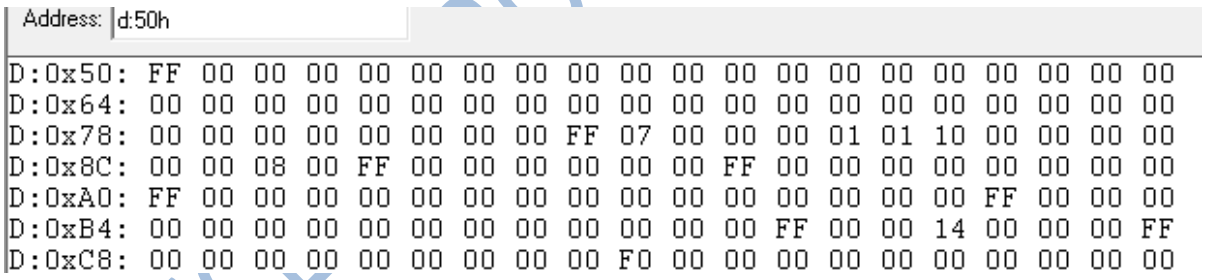
4e.Write an ALP to convert 8 bit Hexadecimal number to Decimal number.

```
org 0000h
sjmp 30h
org 30h
mov a,50h          ; load to accul the hexadecimal number from 50h
mov b,#0ah         ; load B reg=10
div ab             ; divide
mov 61h,b          ; save reminder i.e MSdigit2 of decimal number at 61h
mov b,#0ah
div ab             ; divide quotient at A register by 10
mov 60h,a          ; save quotient at 60h i.e. MSdigit1 of decimal number
mov a,b
swap a
add a,61h
mov 61h,a          ; save reminder at 61h i.e LS digit of decimal number
sjmp $
end
```

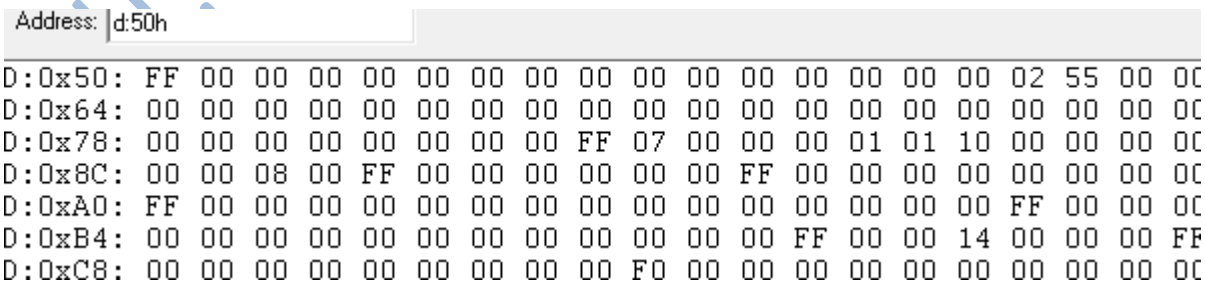
Note: 60h MS digit of decimal number
61h LS digit of decimal number

RESULT:

Before Execution:



After Execution:



4f.Write an ALP to convert 8 bit Decimal number(2 digit) to Hexadecimal number.

```
org 0000h
sjmp 30h
org 30h
mov r0,# 50h            ;pointer to decimal number
mov r2,50h              ;save  the decimal num  at mem.locn to r2 reg
mov a,@r0
anl a,#0f0h             ;mask lower nibble of decimal number
swap a                  ;get  upper nibble of decimal number
mov 0f0h,#10
mul ab                  ;multiply ms digit of the decimal num by 10
mov  r3,0f0h            ;save upper byte at r3 reg
mov r4,a                ;save lower byte at r4 reg.
mov a,r2                ;get the decimal num from r2 reg to acc.
anl a,#0fh              ; mask upper nibble of decimal number
add a,r4          ; add  lower byte
mov 51h,a               ; save result at mem.location 51h
sjmp $
end
```

Note: 50h – Decimal number
51h – Hexadecimal number

Before Execution:

Address: d:50h

```
D:0x50: 48 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x64: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 00 00 00 00
D:0x8C: 00 00 08 00 FF 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00
D:0xA0: FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 00
D:0xB4: 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 14 00 00 00 FF
D:0xC8: 00 00 00 00 00 00 00 00 00 F0 00 00 00 00 00 00 00 00 00 00
```

After Execution:

Address: d:50h

```
D:0x50: 48 30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x64: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 FF 07 00 00 00 01 01 10 00 00 00 00
D:0x8C: 00 00 08 00 FF 00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00
D:0xA0: FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 00
D:0xB4: 00 00 00 00 00 00 00 00 00 00 00 00 FF 00 00 14 00 00 00 FF
D:0xC8: 00 00 00 00 00 00 00 00 40 F0 00 00 00 00 00 00 00 00 00 00
```

IV CYCLE

5a.An example program to demonstrate CALL and RETURN instruction.

```
            org 0000h
            sjmp 30h
            org 30h
            mov a,#34h
            ACALL double
            here:sjmp here
double:     ADD A,#0E0h
            LCALL invert
            ret
            org 5000h
invert:     XRL a,#0ffh
            ret
            end
```

Note: Observe the result in register window(a register)

5b.Write an ALP to generate delay of 0.5ms using software instructions. (generate square wave).*Check the output on logic analyzer window*

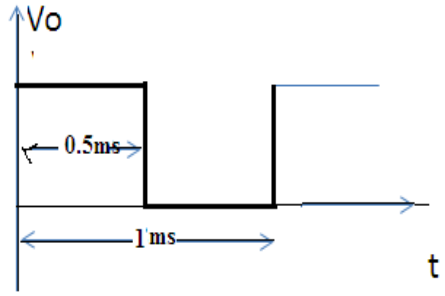
```
        org 0000h
        sjmp start
Start:  clr P1.0              ; p1.0=0(low)
back:   cpl P1.0
        acall delay
        sjmp back
delay:  mov  r4,#0e5h         ; this count generates a delay of  0.5 ms
here:   djnz  r4,here
        ret
        end
```

Delay Calculations:

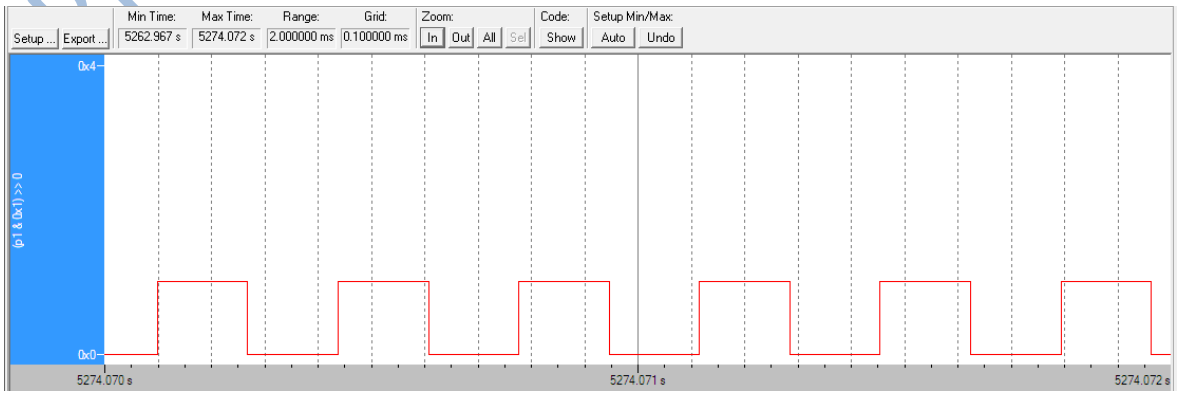
		Machine Cycles	no of times executed
mov r4,#0e5h	(N)	1 m/c	1
here: djnz r4,here		2 m/c	N
ret		2 m/c	1

$(2xN + 2 +1)x1.085x10^{-6} = 0.5x10^{-3}$

N= (229)d= (e5)h



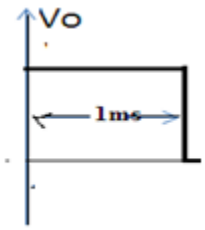
RESULT:



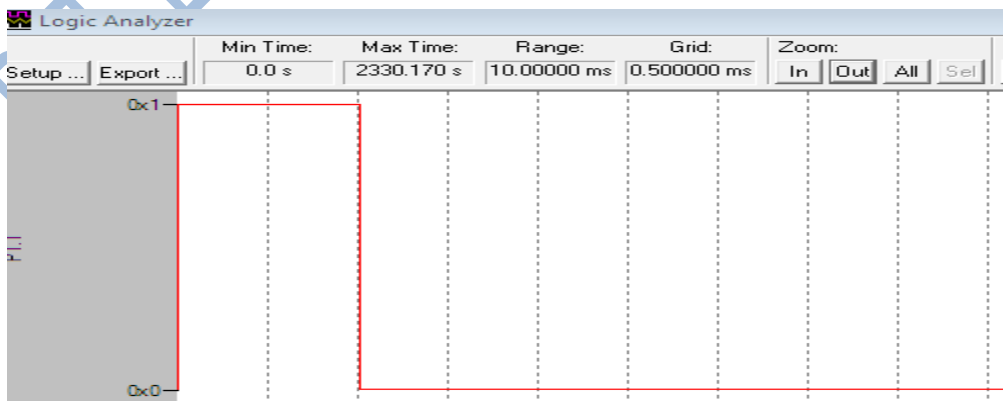
5c.(a)Write an ALP to generate delay of 1ms using on chipTimer1 mode 1
(b)Write an ALP to generate delay of 0.25ms using on chipTimer1 mode 2(Generate a square wave of 2khz) ***Check the output on logic analyzer window***

a). (Generate a delay of 1ms)

```
        org 0000h
        Sjmp 30h
        org 30h
        mov tmod ,#10h          ;configure timer 1 in mode 1
        clr p1.1
        mov th1,#0fch           ;this count gives a delay of 1ms.
        mov tl1,#66h
        cpl p1.1
        Setb tr1                ; start timer
here:   jnb tf1,  here
        clr tf1
        clr tr1
        clr p1.1
 next:  sjmp next
        end
```



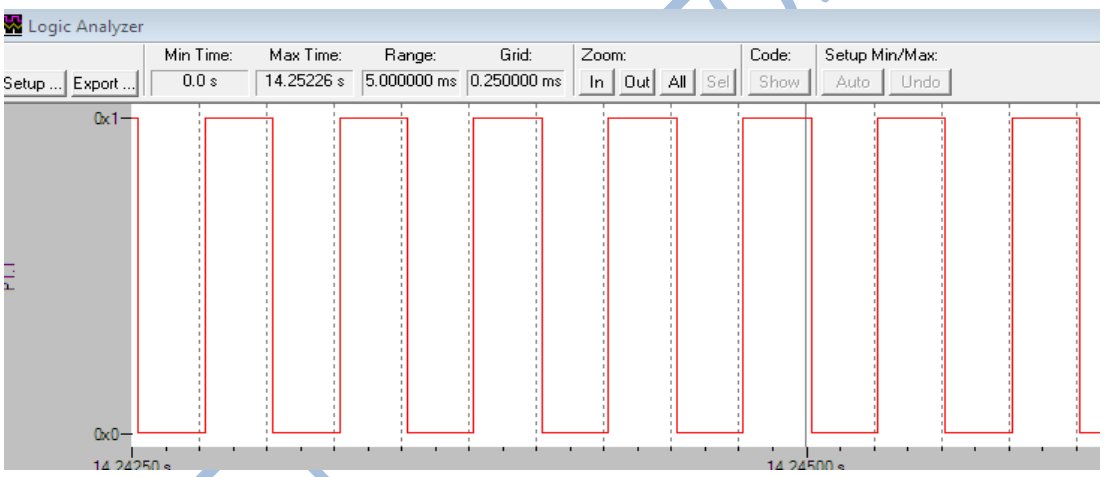
RESULT:



(b) Generate a square wave of 2KHz frequency (each delay of 0.25ms)

```
        org 0000h
        Sjmp start
Start:  mov tmod ,#20h      ;configure timer 1 in mode 2(auto reload)
        mov th1,#1ah        ;this count gives a delay of 1ms.
        clr p1.1
next:   cpl p1.1
        Setb tr1            ; start timer
here:   jnb tf1,  here
        clr tf1
        sjmp next
        end
```

RESULT:

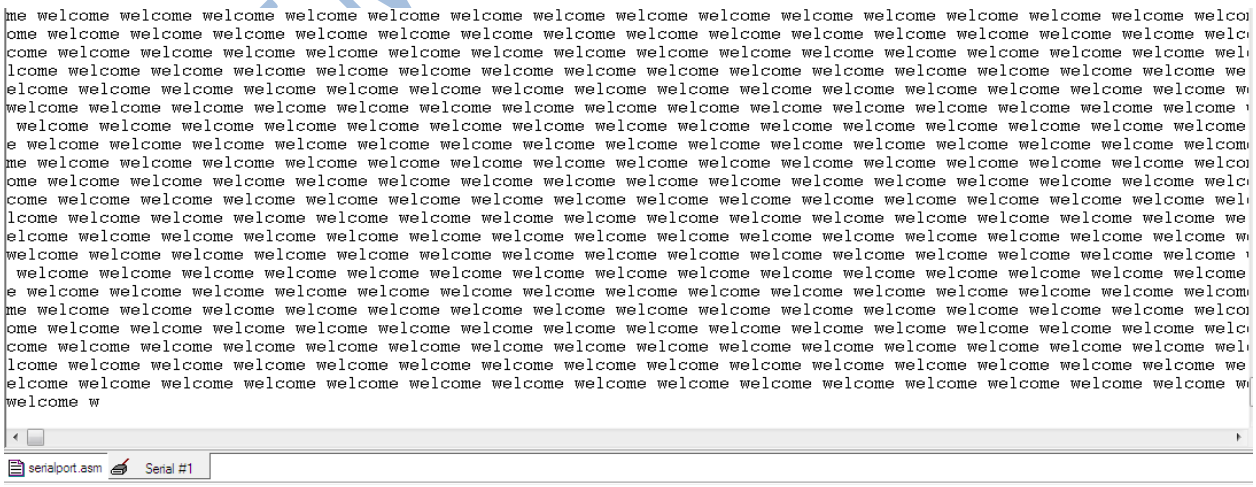


5d.Write an ALP to transfer data serially at 4800 baud rate continuously.

Check the output on serial window1.

```
        org 0000h
        sjmp start
Start:  mov tmod,#20h       ;configure timer 1 for the given baud rate
        mov th1,#0fah
        mov scon,#50h       ; configure UART for mode 1 serial data transfer
        setb tr1
repeat: mov dptr,#mes       ; DPTR points to mem loc which contains message to be
                            ;transmitted
again:  mov a,#00h
        movc a,@a+dptr      ; get a character  and transmit
        jz  skip
        acall trans
        inc dptr
        sjmp again
trans:  mov sbuf,a          ;serial data transfer
here:   jnb ti,here
        clr ti
        ret
skip:   sjmp repeat
        org 100h
mes:    db"welcome ",0h             ; message
        end
```

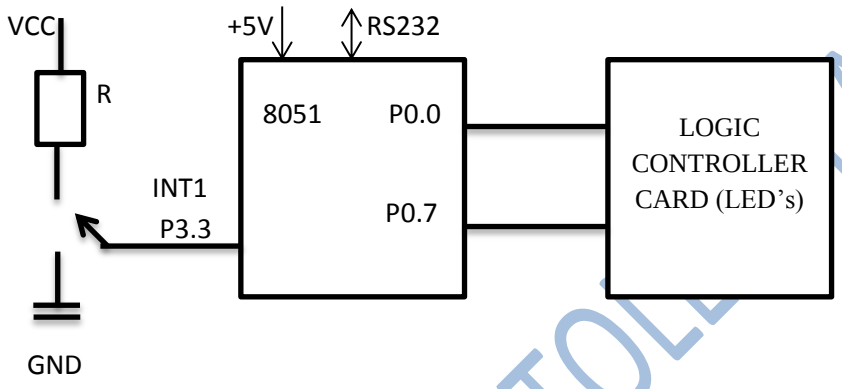
RESULT:



V CYCLE

6a. Interface a simple toggle switch to 8051 and write an ALP to generate an interrupt which switches on an LED (i) continuously as long as switch is on and (ii) only once for a small time when the switch is turned on.

Block Diagram



1.Interrupt with level triggering

```
            org 0000h
            sjmp 0030h          ;main program

----------interrupt service routine for INT1 to turn on/off  led

            org 0013h           ;INT1 vector address
            mov p0,#0aah        ;LED ON/OFF
            mov r4,#255         ;delay for 1 sec
repeat:     mov r5,#255
  next:     mov r6,#04h
    c2:     djnz r6,c2
            djnz r5,next
            djnz r4,repeat
            mov p0,#55h         ;LED ON/OFF
            reti                ; return on interrupt

----------main program for initialization

           org 0030h
            mov p0,#00          ;configure port as output port
            mov ie,#10000100b   ; enable external interrupt 1
here:       sjmp here           ;stay here until interrupted
            end
```

Output for the level triggered interrupt:

2.Interrupt with edge triggering

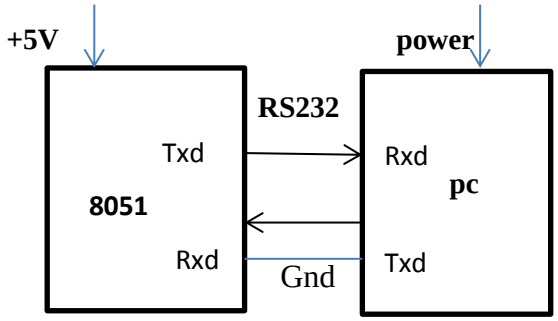
```
            org 0000h
            sjmp 0030h          ;main program
----------interrupt service routine for INT1 to turn on/off led
            org 0013h           ;INT1 vector address
            mov p0,#0aah        ;LED ON/OFF
            mov r4,#255         ;delay for 1 sec
repeat:     mov r5,#255
next:       mov r6,#04h
c2:         djnz r6,c2
            djnz r5,next
            djnz r4,repeat
            mov p0,#55h         ;LED ON/OFF
            reti                ; return on interrupt

----------main program for initialization

            org 0030h
            setb tcon.2         ;make INT1 edge triggered interrupt
            mov p0,#00          ;configure port as output port
            mov ie,#10000100b   ; enable external interrupt 1
here:       sjmp here       ;stay here until interrupted
            end
```

Output for the level triggered interrupt:

6b. Write an ALP to transfer data serially at 4800 baud rate continuously. Check the output on serial window1.



Transmit serially

```
#include<reg51.h>
 void main()
 {
 char A[]="microcontrollerlab,0";
 unsigned char i=0;
 //P3=0x03;
 TMOD=0x20;
 TH1=0xFD;
 SCON=0x50;
 TR1=1;
 while(1)
 {
 while(A[i]!='\0')
 {
 SBUF=A[i];
 while(TI==0);
 TI=0;
 i++;
 }
 i++;
 }
 i=0;
 }
```

Receive serially

```
#include<reg51.h>

sbit rs=P3^7;
sbit en=P3^5;
sbit rw=P3^6;

void delay(unsigned int time)
{ unsigned int i,j;
```

```
        for(i=0;i<time;i++)
        for(j=0;j<5;j++);
        }
void lcdcmd(unsigned char value)
{
P2=value;
rs=0;
rw=0;
en=1;
delay(50);
en=0;
delay(50);
}
void display(unsigned char value)
{
P2=value;
rs=1;
rw=0;
en=1;
delay(50);
en=0;
delay(50);

}
void lcdinit(void)
{
P2=0x00;
P3=0x03;
delay(1500);
display(0x30);
delay(450);
display(0x30);
delay(300);
display(0x30);
delay(650);
lcdcmd(0x38);
delay(50);
lcdcmd(0x0E);
delay(50);
lcdcmd(0x01);
delay(50);
lcdcmd(0x06);
delay(50);
lcdcmd(0x00);
delay(50);
}

void main()
{
```

```
  char data1;
  lcdinit();

TMOD=0x20;
  TH1=0xFD;
  SCON =0x50;
  TR1=1;
  while(1)
  {
  while(SBUF!=0x0D)
  {
  while(RI==0);
  data1=SBUF;
  RI=0;
  display(data1);
  delay(50);
  }
  }
  }
```

Serial communication between the computer and the **microcontroller** follows the below steps:

Step1: Set the serial port mode.

Step2: Set the serial port baud rate.

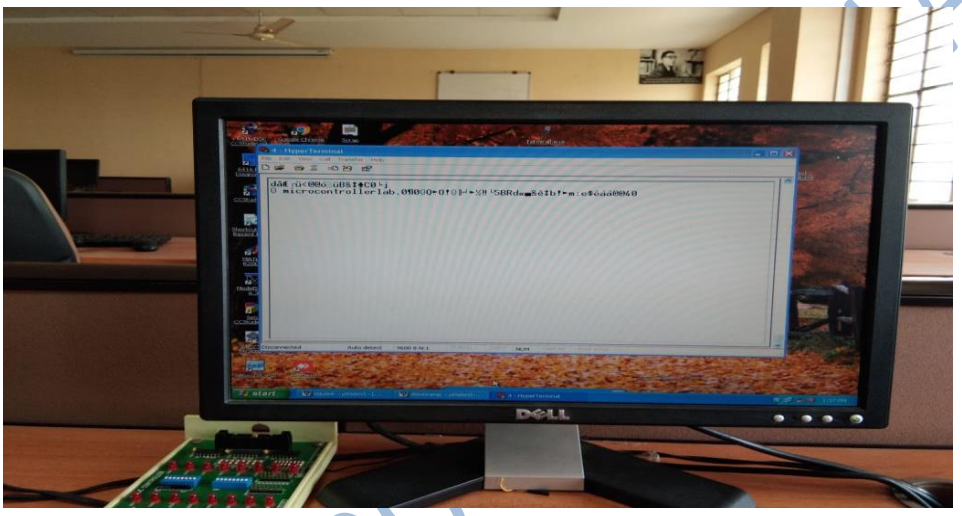
Step3: Write and read serial port.

After step 1 and 2, the serial port can be used for transmission and reception of data. Hyper terminal is used for the reception and transmission of data through RS232.

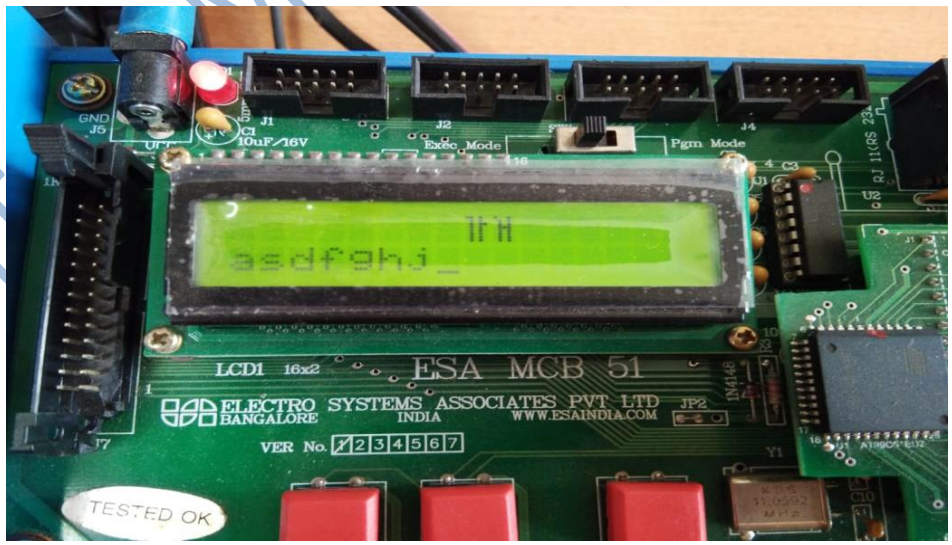
- To open Hyper Terminal, go to Start Menu, Programs and Accessories.
- Click Communications.
- Select Hyper Terminal.
- Go to File.
- Click new connection.
- Give name to connection.
- Select Baud rate 9600bps.
- Parity as none, Data bits 8, Flow control none Stop bit 1.

In the program of serial transmission, the character transmitted through SBUF of the controller and is displayed on the hyper terminal of PC. While, in case of serial reception, data is entered on the hyper terminal and it is serially received through the SBUF of ***microcontroller***. Inorder to view the output on the microcontroller, in case of serial reception, **LCD** interfacing has to be done.

Output for the serial Transmission seen on Hyper Terminal Window:

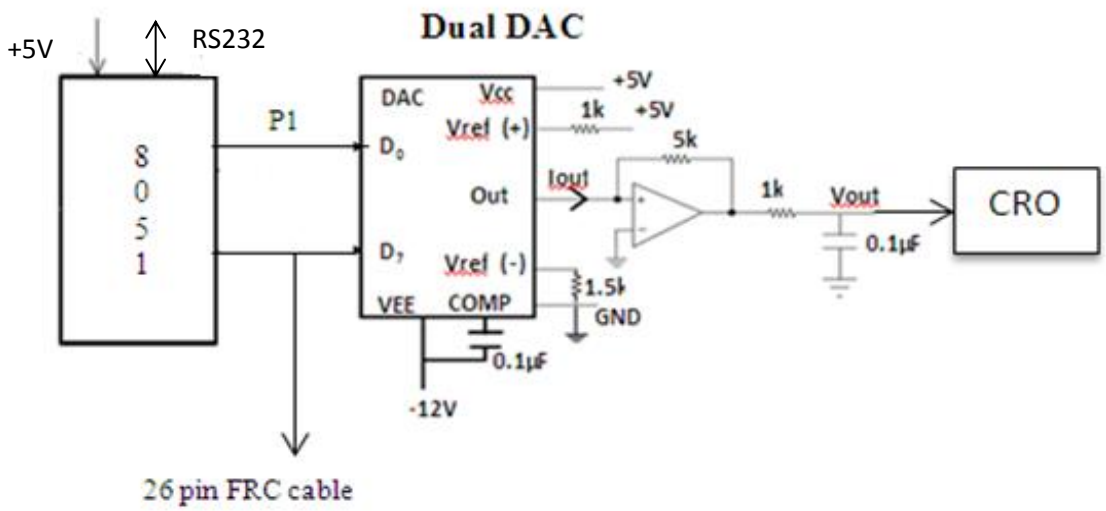


Serial receiving output seen on LCD display:



6c. Write ALPs to generate waveforms using DAC interface.

Block Diagram



1. Sine wave

Calculation to generate the sine wave:

Angle in degrees	Sinθ	Vout =[2.5v+(2.5Sinθ)]	Values to DAC (decimal) Vout*256/5v
0	0	2.5v	128
10	0.1736	2.934v	151
.	.	.	.
.	.	.	.
350.	-0.17.	2.06	106
360	0	2.5v	128

```
        org 0000h
        sjmp 30h
        org 30h
        mov p1,#00                      ; configure p1 as o/p port
        again:mov dptr, #table
        mov r2,#36
        back: clr a
        movc a, @a+dptr                 ; send data from table to DAC to
                                          generate   sine wave
        mov p1,a
        inc dptr
        djnz r2,back
        sjmp again
        org 300h
table:  db 128,151,172,192,210,226,239,248,254,255,254,248,239,226,210
        db 192,172,150,128,106,84,64,46,30,17,8,2,0,2,8,17,30,46,64,84,106

        end
```

Output for the Sin wave:

2.square wave

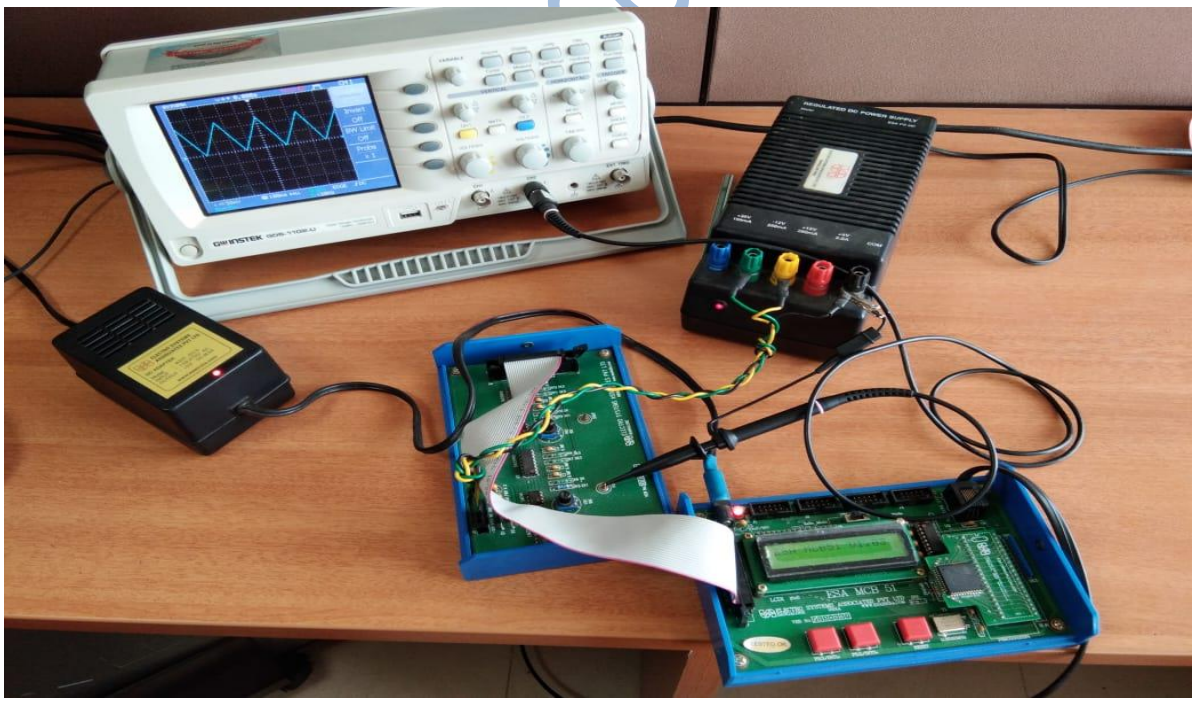
```
        org 0000h
        sjmp 0030h
        org 0030h
        mov p1,#00h                ; configure p1 as o/p port
       //mov p2,#00h
        repeat: clr a
        mov p1,a                   ; send 00 an p1
        call delay                 ; delay
        mov a,#0ffh                ; send FFh(high) an p1
        mov p1,a
        call delay                 ; delay
        sjmp repeat                ; repeat to generate square wave
delay:  mov r1,#05h
back1:  mov r2,#0e5h
 back:  djnz r2,back
        djnz r1,back1
        ret
        end
```

Output for the Square wave:

3.Triangle wave

```
        org 0000h
        sjmp 030h
        org 0030h
        mov p1,#00h                ; configure p1 as o/p port
repeat: mov a,#00h                 ; generate triangular wave an port p1
  up:   mov p1,a
        inc a
        call delay
        cjne a,#0ffh,up
        dec a                      ; generate up ramp
down:   mov p1,a
        dec a
        acall delay
        cjne a,#0ffh,down          ; generate down ramp
        sjmp repeat
delay:  mov r3,#0e0h               ; delay between steps
 back:  djnz r3,back
        ret
        end
```

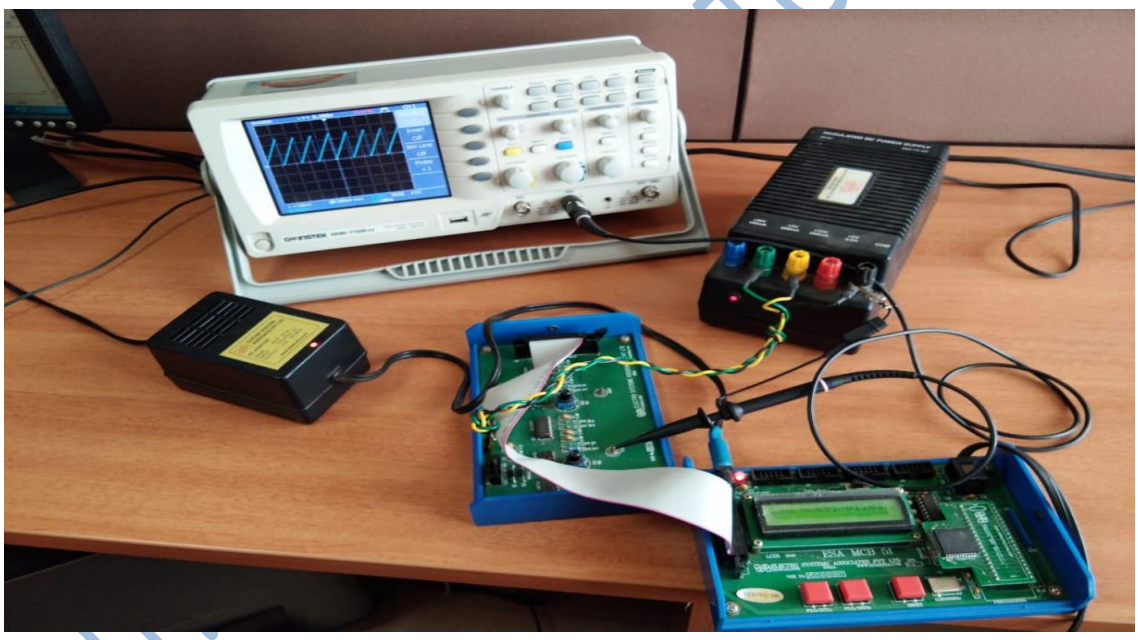
Output for the Triangle wave:



4.Up ramp

```
        org 0000h
        sjmp 030h
        org 0030h
        mov p1,#00h          ; configure p1 as o/p port
        clr a                ; A=0
again:  mov p1,a             ; o/p increment value of A
        inc a
        acall delay          ; delay between steps
        sjmp again           ; repeat forever
delay:  mov r3,#0e0h         ;delay program
back:   djnz r3,back
        ret
        end
```

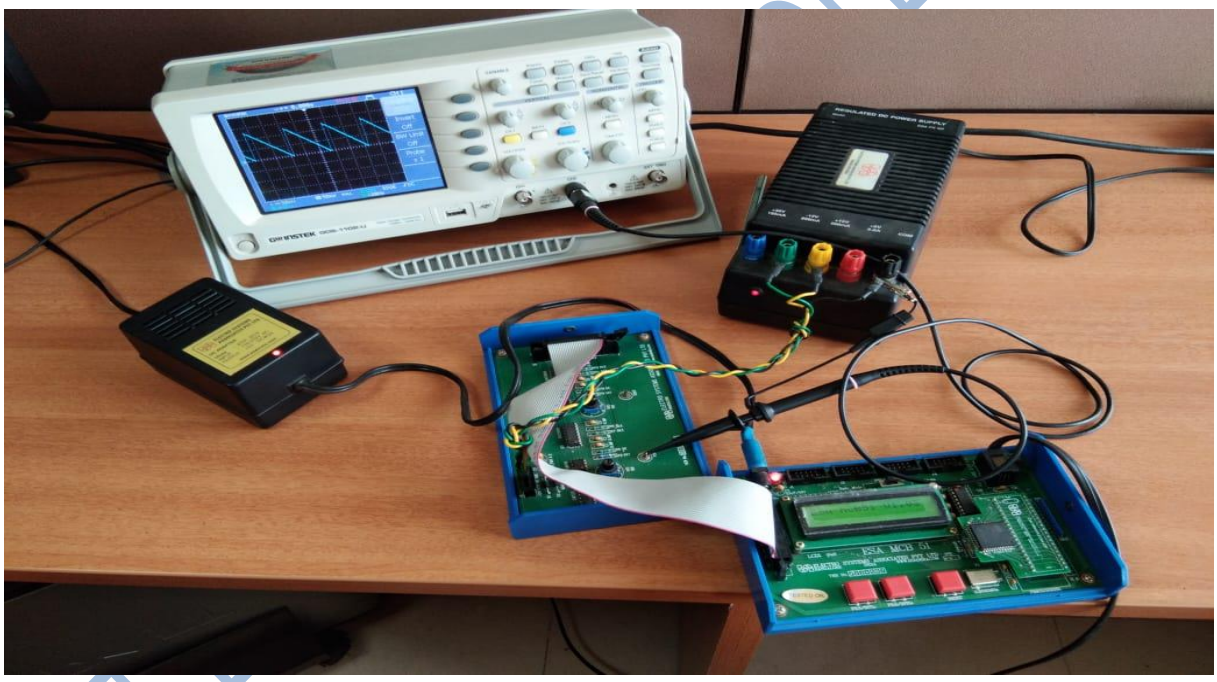
Output for the Up ramp:



4.Down_ramp

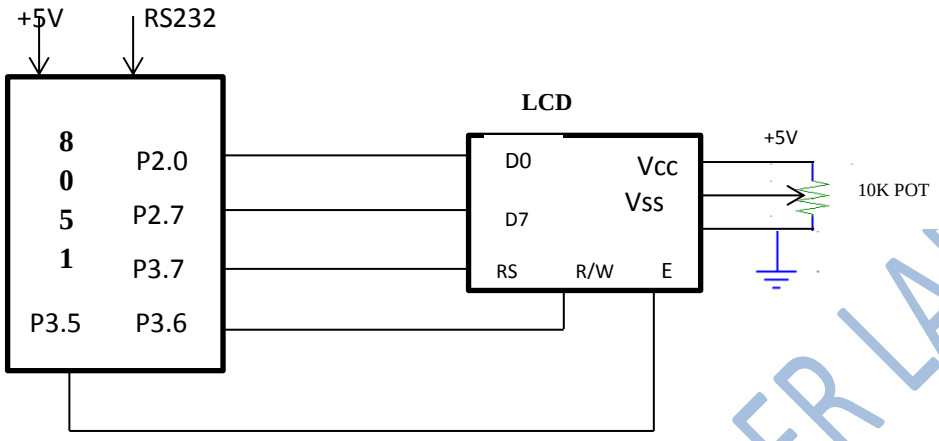
```
        org 0000h
        sjmp 030h
        org 0030h
        mov p1,#00h          ; configure p1 as o/p port
        mov a,#0ffh          ; A= 0ffh
again:  mov p1,a             ; output high at port 1
        dec a
        acall delay          ; decrement Accumulator
        sjmp again           ; Delay between steps
delay:  mov r3,#0e0h
back:   djnz r3,back
        ret
        end
```

Output for the Down ramp:



6d.Write ALP to interface an LCD display and to display a message on it.

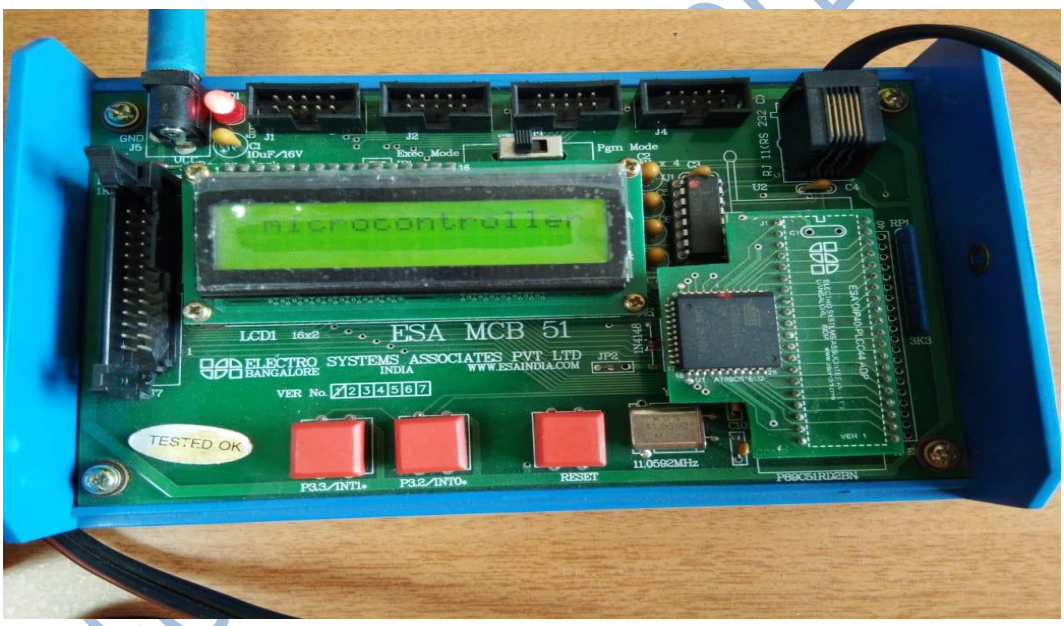
Block Diagram




```
        org 0000h
        sjmp 0030h
        org 0030h
        mov dptr,#mcomm
c1:     clr a
        movc a,@a+dptr
        acall commwrt           ; call command subroutine
        acall delay             ; give LCD some time
        jz send_dat
        inc dptr
        sjmp c1
send_dat:   mov dptr,#mdata
d1:     clr a
        movc a,@a+dptr
        acall datawrt           ; call data with  subroutine
        acall delay             ; give LCD some time
        inc dptr
        jz again
        sjmp d1
again:  sjmp again              ;stay here
commwrt:    mov p2,a            ; send command to lcd through p2
        clr p3.7                ; RS=0 for command
        clr p3.6                ; R/W= for write
        setb p3.5               ; E=1
        acall delay             ; delay
        clr p3.5                ; E=0 for H to L
        ret
datawrt:  mov p2,a              ; send data to p2
        setb  p3.7              ; RS=1 for data
        clr p3.6                ; R/W=0 for write
```

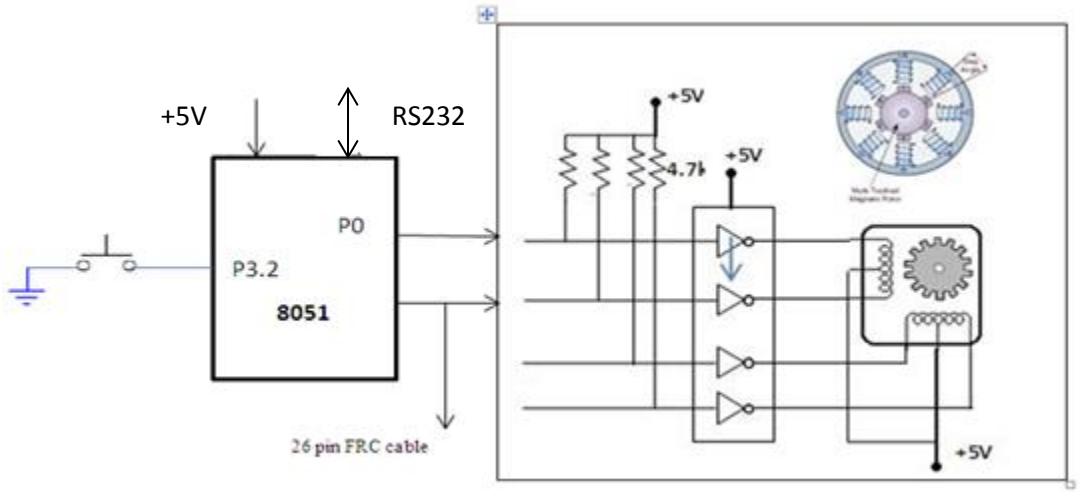
```
        setb p3.5                       ; E=1
        acall delay                     ; delay
        clr p3.5                        ; E=0 for H to L pulse
        ret
delay:  mov r3,#50                      ; Long delay
here2:  mov r4,#255
here:   djnz r4,here
        djnz r3,here2
        ret
        org 0300h
mcomm: db 38h,0eh,01h,06h,00h,0         ; commands and null
mdata: db " microcontroller "           ; data and null
        end
```

Output seen on the LCD Display:



6e.Write ALP to interface a Stepper Motor to 8051 to rotate the motor.

Block Diagram



P0.0 to P0.3 connections

i.continous rotation

```
        org 0000h
        sjmp 30h
        org 0030h
        setb p3.2          ; p3.2 configured as i/p port
        mov p0,#00h        ; port 0 configured as o/p port
        mov a,#88h         ; load step sequence
cw:     jnb p3.2, ccw      ; check status of switch, if switch is not pressed
                             p3.2=1
        mov p0,a           ; issue sequence to motor
        rr a               ; for rotate right clockwise
        acall delay
        acall delay
        sjmp cw
ccw:    rl a               ; if p3.2=0(switches pressed)
        acall delay        ; issue sequence to motor
        acall delay
        mov p0,a           ; for rotateing counter clockwise
        sjmp cw
delay:  mov r4,#07h        ; repeat
repeat: mov r5,#0ffh       ; delay between steps
next:   djnz r5,next
        djnz r4,repeat
        ret
        end
```

ii.90º rotation

```
        org 0000h
        sjmp 30h
        org 0030h
        mov p0,#00h          ; port 0 configured as o/p port
        mov a,#88h           ; Load step sequence.
        mov r0,#50           ; for 90º degree of rotation / step angle = 90 / 1.8 =50)
back:   rr a
        mov p0,a             ; issue sequence to motor
        acall delay          ; wait
        acall delay
        djnz r0,back
again:  sjmp again           ; stay here
delay:  mov r4,#07h
repeat: mov r5,#0ffh
next:   djnz r5,next
        djnz r4,repeat
        ret
        end
```

Connection for the stepper motor interface: