Microcontrollers Lab Manual

February 2016

Prepared by:
Mrs. Ashwini K B
Mrs. Roopa S
Ms. Sucheta N Angadi

Reviewed by:
Mr. Lokhanathan M S
Assistant Professor

Approved by:
Prof. V C Kumar
Professor & Head,
Dept. of EEE
I. PROGRAMMING

3. Counters.
4. Boolean & Logical Instructions (Bit manipulations).
5. Conditional CALL & RETURN.
6. Code conversion: BCD – ASCII; ASCII – Decimal; Decimal - ASCII;
7. HEX - Decimal and Decimal - HEX.
8. Programs to generate delay, Programs using serial port and on-Chip timer/Counter.

Note: Programming exercise is to be done on both 8051 & MSP430.

II. INTERFACING

Write C programs to interface 8051 chip to Interfacing modules to develop single chip solutions.

9. Simple Calculator using 6 digit seven segment displays and Hex Keyboard interface to 8051.
10. Alphanumeric LCD panel and Hex keypad input interface to 8051.
11. External ADC and Temperature control interface to 8051.
12. Generate different waveforms Sine, Square, Triangular, Ramp etc. using DAC interface to 8051; change the frequency and amplitude.
13. Stepper and DC motor control interface to 8051.
14. Elevator interface to 8051.
## Index Page

<table>
<thead>
<tr>
<th>Sl. No</th>
<th>Name of the Experiment</th>
<th>Date</th>
<th>Manual Marks (Max. 25)</th>
<th>Record Marks (Max. 10)</th>
<th>Signature (Student)</th>
<th>Signature (Faculty)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Conduction</td>
<td>Repetition</td>
<td>Submission of Record</td>
<td></td>
</tr>
</tbody>
</table>

**PART-A1: Assembly language programming using 8051**

01 Data Transfer Programming
02 Arithmetic Instruction Programming
03 Boolean & Logical Instructions Programming
04 Counter Programming
05 Code Conversion Programming
06 Serial Communication Programming

**PART-B: 8051 Interfacing programs**

07 C program for waveform generation using DAC
08 C program to motor Interfacing
09 C program to LCD and keypad Interfacing
10 C program to interface Elevator
11 program to interface Calculator using Keyboard
12 C program to interface Temperature sensor Program for temperature sensor.

**PART-A2: Programming using MSP430**

13 Data Transfer Programming, Arithmetic Instruction Programming
14 Boolean & Logical Instructions Programming,
15 Counter Programming, Code Conversion Programming

**Average**
OBJECTIVES AND OUTCOMES OF THE COURSE

OBJECTIVES

The objectives of the course are:

- To introduce the basics of microcontroller and its applications.
- To provide in depth knowledge of 8051 and MSP 430 assembly language programming.
- To expertise working with Keil compiler and embedded C programming.
- To impart the I/O interfacing concepts for developing real time embedded systems.
- To encourage the students in building real time applications.

OUTCOMES

Upon completion of the lab course, students will be able to:

- Familiarize with the assembly level and embedded C programming using 8051.
- Familiarize with the assembly level programming using low powered MSP430.
- Familiarize with the Keil µVision-3/4 and IAR Embedded Workbench tools.
- Design circuits for various applications using microcontrollers.
- Apply the concepts on real- time applications.
GENERAL INSTRUCTIONS TO STUDENTS

1. Students should come with thorough preparation for the experiment to be conducted.

2. Students should take prior permission from the concerned faculty before availing the leave.

3. Students should come with formals. And to be present on time in the laboratory.

4. Students will not be permitted to attend the laboratory unless they bring the practical record fully completed in all respects pertaining to the experiment conducted in the previous class.

5. Students will be permitted to attend laboratory unless they bring the observation book fully completed in all respects pertaining to the experiment conducted in the present class.

6. They should obtain the signature of the staff-in-charge in the observation book after completing each experiment.

7. Practical record and observation book should be maintained neatly.
# TABLE OF CONTENTS

## PART-A1: Assembly language programming using 8051

<table>
<thead>
<tr>
<th>Sl. No.</th>
<th>Name of the Design</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Data Transfer Programming</td>
<td>05</td>
</tr>
<tr>
<td>2.</td>
<td>Arithmetic Instruction Programming</td>
<td>09</td>
</tr>
<tr>
<td>3.</td>
<td>Boolean &amp; Logical Instructions Programming</td>
<td>13</td>
</tr>
<tr>
<td>4.</td>
<td>Counter Programming</td>
<td>16</td>
</tr>
<tr>
<td>5.</td>
<td>Code Conversion Programming</td>
<td>17</td>
</tr>
<tr>
<td>6.</td>
<td>Serial Communication Programming</td>
<td>19</td>
</tr>
</tbody>
</table>

## PART-B: 8051 Interfacing programs

<table>
<thead>
<tr>
<th>Sl. No.</th>
<th>Name of the Design</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.</td>
<td>C program for waveform generation using DAC</td>
<td>21</td>
</tr>
<tr>
<td>8.</td>
<td>C program to motor Interfacing</td>
<td>25</td>
</tr>
<tr>
<td>9.</td>
<td>C program to LCD and keypad Interfacing</td>
<td>29</td>
</tr>
<tr>
<td>10.</td>
<td>C program to interface Elevator</td>
<td>31</td>
</tr>
<tr>
<td>11.</td>
<td>C program to interface Calculator using Keyboard</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>and 7-segment display.</td>
<td></td>
</tr>
<tr>
<td>12.</td>
<td>C program to interface Temperature Sensor Program for</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>temperature sensor.</td>
<td></td>
</tr>
</tbody>
</table>
### PART-A2: Programming using MSP430

<table>
<thead>
<tr>
<th>Sl. No.</th>
<th>Name of the Design</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.</td>
<td>Data Transfer &amp; Arithmetic Instruction Programming</td>
<td>43</td>
</tr>
<tr>
<td>14.</td>
<td>Boolean &amp; Logical Instructions Programming</td>
<td>49</td>
</tr>
<tr>
<td>15.</td>
<td>Counter &amp; Code Conversion Programming</td>
<td>51</td>
</tr>
</tbody>
</table>

- **Additional Programs**: 57
- **References**: 59
- **Viva Questions**: 60
- **Question Bank**: 63
- **Annexure**: 65
INTRODUCTION

Earlier to Microcontrollers, Microprocessors were greatly used for each and every purpose. Microprocessors were containing ALU, general purpose register, stack pointer, program counter, clock counter and so many other features which the today’s Microcontroller also possesses. But the difference between them exists with respect to the number of instructions, access times, size, reliability, PCB size and so on. Microprocessor contains large instruction set called as CISC processor whereas Microcontroller contains less number of instructions and is called as RISC processor. The access time is less in case of microcontrollers compared to microprocessors and the PCB size reduces in case of microcontrollers.

There are many versions of microcontrollers 8051, 80528751, AT8951 from Atmel Corporation and many more. In this manual we will study about the 8051 architecture, its features, programming and interfacing.

MCS 8051 is an 8-bit single chip microcontroller with many built-in functions and is the core for all MCS-51 devices.

The main features of the 8051 core are:

- Operates with single Power Supply +5V.
- 8-bit CPU optimized for control applications.
- 16-bit program counter (PC) and 16-bit data pointer (DPTR).
- 8-bit program status word (PSW).
- 8-bit stack pointer (SP).
- 4K Bytes of On-Chip Program Memory (Internal ROM or EPROM).
- 128 bytes of On-Chip Data Memory (Internal RAM):
  - Four Register Banks, each containing 8 registers (R0 to R7) [Total 32 reg]
  - 16-bytes of bit addressable memory.
  - 80 bytes of general-purpose data memory (Scratch Pad Area).
- Special Function Registers (SFR) to configure/operate microcontroller.
- 32 bit bi-directional I/O Lines (4 ports P0 to P3).
- Two 16-bit timers/counters (T0 and T1).
- Full duplex UART (Universal Asynchronous Receiver/Transmitter).
- On-Chip oscillator and clock circuitry.
**STEPS TO CREATE AND COMPILE Keil µVision-3/4 PROJECT:**

1. Double Click on the µVision3/4 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 or AT89C51RD2 as per the board). On clicking ‘OK’, the following option is displayed. Choose ‘No’.

```
Copy Standard 8051 Startup Code to Project Folder and Add File to Project?

Yes  No
```

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->Manage->Components, Environment 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.

![Project Workspace](image)
5. Set the Target options using -> Project – Options for Target opens the µVision2 Options for Target – Target configuration dialog. Set the Xtal (Crystal frequency) frequency as 11.0592 MHz, and also the Options for Target – Debug – use either Simulator / Keil Monitor- 51 driver.

6. 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.

7. Build the project; using Project -> Build Project. µVision 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, to build go to Debug mode.

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

9. 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, and step into, run till cursor).

10. 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.
11. **Note:** To access data RAM area type address as D: 0020h. Similarly to access the DPTR region (XRAM-present on chip in AT89C51ED2) say 9000h location type in X: 09000H.
Experiment no. 1 : Data Transfer Programming

a. Write an assembly language program to transfer $N = ____$ bytes of data from location $A: ____h$ to location $B: ____h$.

Let $N = 05h$, $A: 30h$ $B: 40h$

```assembly
mov r0,#30h //source address
mov r1,#40h //destination address
mov r7,#05h //Number of bytes to be moved
back: mov a,@r0
mov @r1,a
inc r0
inc r1
djnz r7,back //repeat till all data transferred
end
```

Result:

Before Execution:

![Before Execution](image1)

After Execution:

![After Execution](image2)
b. Write an assembly language program to exchange $N = ____h$ bytes of data at location A: ____h and at location B: ____h.

Let $N = 05h$  
A: 30h  
B: 40h  

c. mov r0,#30h  
   //source address  
mov r1,#40h  
   //destination address  
mov r7,#05h  
   //count, the number of data to be exchanged  

back:  
mov a,@r0  
mov r4,a  
mov a,@r1  
mov @r0,a  
mov a,r4  
mov @r1,a  
inc r0  
inc r1  
djnz r7,back  
end

Result:
C. Write an assembly language program to find the largest element in a given array of \( N = \_\_\_ \) bytes at location 4000h. Store the largest element at location 4062h.

Let \( N = 06h \)

\[
\begin{align*}
\text{mov} & \ r3, #6 & & \text{//length of the array} \\
\text{mov} & \ dptr, #4000H & & \text{//starting address of array} \\
\text{movx} & \ a, @dptr \\
\text{mov} & \ r1, a \\
\text{nextbyte: inc dptr} \\
\text{movx} & \ a, @dptr & & \text{//reset borrow flag} \\
\text{clr} & \ c \\
\text{mov} & \ r2, a & & \text{//next number in the array} \\
\text{subb} & \ a, r1 & & \text{//other Num-Prev largest no.} \\
\text{jc} & \ skip & & \text{// JNC FOR SMALLEST ELEMENT} \\
\text{mov} & \ a, r2 & & \text{//update larger number in r1} \\
\text{mov} & \ r1, a \\
\text{skip: djnz r3, nextbyte} \\
\text{mov} & \ dptr, #4062H & & \text{//location of the result-4062h} \\
\text{mov} & \ a, r1 & & \text{//largest number} \\
\text{movx} & \ @dptr, a & & \text{//store at #4062H} \\
\text{end}
\end{align*}
\]

Result:

Before Execution:

<table>
<thead>
<tr>
<th>Memory 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address: x:4000h</td>
</tr>
<tr>
<td>X:0x004000: 01 02 03 FF 66 00</td>
</tr>
<tr>
<td>X:0x004025: 00 00 00 00 00 00</td>
</tr>
</tbody>
</table>

After Execution:

<table>
<thead>
<tr>
<th>Memory 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address: x:4062h</td>
</tr>
<tr>
<td>X:0x004062: FF 00 00 00 00 00</td>
</tr>
<tr>
<td>X:0x004087: 00 00 00 00 00 00</td>
</tr>
</tbody>
</table>
d. Write an assembly language program to sort an array of $N = \_\_\_h$ bytes of data in ascending/descending order stored from location 9000h. (Using bubble sort algorithm)

Let $N = 06h$

```
mov r0,#05H //count (N-1) array size = N
loop1: mov dptr, #9000h //array stored from address 9000h
        mov r1,#05h //initialize exchange counter
loop2: movx a, @dptr //get number from array and store in B register
        mov b, a
        inc dptr
        movx a, @dptr //next number in the array
        clr c //reset borrow flag
        mov r2, a //store in R2
        subb a, b //2nd-1st No, since no compare instruction in 8051
        jnc noexchg //JC - FOR DESCENDING ORDER
        mov a, b //exchange the 2 nos in the array
        movx @dptr,a //exchange the 2 nos in the array
        dec dpl //DEC DPTR - instruction not present
        mov a, r2
        movx @dptr,a
        inc dptr
noexchg: djnz r1,loop2 //decrement compare counter
         djnz r0,loop1 //decrement pass counter
end
```

Result:

**Before Execution:**

<table>
<thead>
<tr>
<th>Memory 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address: 93000h</td>
</tr>
<tr>
<td>X: 0x00900C: 66 55 44 33 22 11 00 00</td>
</tr>
<tr>
<td>X: 0x009025: 00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>X: 0x00904A: 00 00 00 00 00 00 00 00</td>
</tr>
</tbody>
</table>

**After Execution:** (Ascending order)

<table>
<thead>
<tr>
<th>Memory 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address: 93000h</td>
</tr>
<tr>
<td>X: 0x00900C: 11 22 33 44 55 66 00 00</td>
</tr>
<tr>
<td>X: 0x009025: 00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>X: 0x00904A: 00 00 00 00 00 00 00 00</td>
</tr>
</tbody>
</table>

**Note:**

Analyze the bubble sort algorithm for the given data. Also try with different sorting algorithms.
Experiment no. 2: Arithmetic Instruction Programming

a. Write an assembly language program to perform the addition of two 16-bit numbers.

```assembly
mov r0,#34h //lower nibble of No.1
mov r1,#12h //higher nibble of No.1
mov r2,#0dch //lower nibble of No.2
mov r3,#0feh //higher nibble of No.2
clr c
mov a,r0
add a,r2
mov 22h,a
mov a,r1
addc a,r3
mov 21h,a
mov 00h,c
end
```

Result:

Input: 

Output: 
b. Write an assembly language program to perform the subtraction of two 16-bit numbers.

```
mov r0,#0dch     //lower nibble of No.1
mov r1,#0feh     //higher nibble of No.1
mov r2,#34h      //lower nibble of No.2
mov r3,#12h      //higher nibble of No.2
clr c            //
mov a,r0
subb a,r2
mov a,r1
subb a,r3
mov 22h,a
mov 21h,a
mov 00h,c
end
```

**Result:**

```
Memory 3

<table>
<thead>
<tr>
<th>Address:</th>
<th>20h</th>
</tr>
</thead>
<tbody>
<tr>
<td>D:0x20h</td>
<td>00</td>
</tr>
<tr>
<td>D:0x47h</td>
<td>00</td>
</tr>
</tbody>
</table>
```

**Note:** Try with different data. Ex: (Smaller number) – (larger number).
c. Write an assembly language program to perform the multiplication of two 16-bit numbers.

```
mov r0,#34h  // 5678*1234
mov r1,#12h
mov r2,#78h
mov r3,#56h
mov a,r0
mov b,r2
mul ab
mov 33h,a
mov r4,b
mov a,r0
mov b,r3
mul ab
add a,r4
mov r5,a
mov a,b
addc a,#00h
mov r6,a
mov a,r1
mov b,r2
mul ab
add a,r5
mov 32h,a
mov a,b
addc a,r6
mov 00h,c
mov r7,a
mov a,r3
mov b,r1
mul ab
add a,r7
mov 31h,a
mov a,b
addc a,20h
mov 30h,a
end
```

Result:

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x30</td>
<td>06 26 00 00 00 00 00 00</td>
</tr>
<tr>
<td>0x31</td>
<td>00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>0x32</td>
<td>00 00 00 00 00 00 00 00</td>
</tr>
<tr>
<td>0x33</td>
<td>00 00 00 00 00 00 00 00</td>
</tr>
</tbody>
</table>

Note: Write the logic of the program. Try with some other logic.
d. Write an assembly language program to find the square of a given number \( N \).

Let \( N = 05 \)

```
mov a,#05  // a=N=05
mov b,a
mul ab
mov 30h,a  // result is stored in 30h and 31h
mov 31h,b
end
```

**Result:**

Input: Output:

---

e. Write an assembly language program to find the cube of a given number.

```
mov r0,#0fh  // r0 = given number to find the cube of it.
mov a,r0
mov b,r0
mul ab
mov r1,b
mov b,r0
mul ab
mov 32h,a
mov r2,b
mov a,r1
mov b,r0
mul ab
add a,r2
mov 31h,a
mov a,b
addc a,#00h
mov 30h,a  //result is stored in 30h, 31h, 32h
end
```

**Result:**

Input: Output:
Experiment no.3: Boolean & Logical Instructions Programming

a. Write an ALP to compare two eight bit numbers NUM1 and NUM2 stored in external memory locations 8000h and 8001h respectively. Reflect your result as: If NUM1<NUM2, SET LSB of data RAM location 2FH (bit address 78H). If NUM1>NUM2, SET MSB of location 2FH (bit address 7FH). If NUM1 = NUM2, then Clear both LSB & MSB of bit addressable memory location 2FH.

```
mov dptr,#8000h
movx a,@dptr
mov r0,a
inc dptr
movx a,@dptr
clr c
sub a,r0
jz equal
jnc small
setb 7fh
sjmp end1
small: setb 78h
sjmp end1
equal: clr 78h
clr 7fh
end1:
end
```

Result:
1) Before Execution: X: 8000h = & X: 8001 =
After Execution: D: 02FH =
2) Before Execution: X: 8000h = & X: 8001 =
After Execution: D: 02FH =
3) Before Execution: X: 8000h = & X: 8001 =
After Execution: D: 02FH =
b. Write an assembly language program to count number of ones and zeros in a eight bit number.

```
mov  r1,#00h  // to count number of 0s
mov  r2,#00h  // to count number of 1s
mov  r7,#08h  // counter for 8-bits
mov  a,#97h   // data to count number of 1s and 0s
again:  rlc a
        jc  next
        inc r1
        sjmp here
next:   inc r2
here:   djnz  r7,again
end
```

Result:

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Number of zero’s = r2 =</td>
</tr>
<tr>
<td></td>
<td>Number of one’s = r1 =</td>
</tr>
</tbody>
</table>

c. Write an assembly language program to find whether given eight bit number is odd or even. If odd store 00h in accumulator. If even store FFh in accumulator.

```
mov  a,20h   // 20h=given number, to find is it even or odd
        jb  acc.0,odd  //jump if direct bit is set i.e., if lower bit is 1 then number is odd
mov  a,#0FFh
        sjmp  ext
odd:   mov  a,#00h
ext:   end
```

Result:

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>20h:</td>
<td>a:</td>
</tr>
</tbody>
</table>
d. Write an assembly language program to perform logical operations AND, OR, XOR on two eight bit numbers stored in internal RAM locations 21h, 22h.

```assembly
mov a, 21h           // do not use #, as data ram 21h is to be accessed
anl a, 22h           // logical and operation
mov 30h, a           // and operation result stored in 30h
mov a, 21h
orl a, 22h           // logical or operation
mov 31h, a           // or operation result stored in 31h
mov a, 21h
xrl a, 22h           // logical xor operation
mov 32h, a           // xor operation result stored in 32h
end
```

Result:

Before Execution: D: 21H = 22H =

After Execution:

D: 030H = // AND operation
D: 031H = // OR operation
D: 032H = // XOR operation


e. Write a Program to check whether given number is palindrome or not. If palindrome store FFh in accumulator else store 00h in accumulator.

```assembly
mov 30h, #81h
mov r0, 30h
mov r1, #08h
mov 31h, #00h
clr c
back:  mov a, 30h
rlc a
mov 30h, a
mov a, 31h
rrc a
mov 31h, a
djnz r1, back
cjne a, #00h, npal
mov a, #0ffh
jmp next
npal:  mov a, #00h
next:   end
```

Result:

Input: __________________________ Output: __________________________
Experiment no.4 : Counter Programming

a. Write an assembly language program to implement (display) an eight bit UP/DOWN binary (hex) counter on watch window.

mov a, #00
back: acall delay
inc a
jnz back
here: sjmp here
delay: mov r1, #0ffh
decr1: mov r2, #0ffh
decr: mov r3, #0ffh
djnz r3, $
djnz r2, decr
djnz r1, decr1
ret
end

RESULT: Accumulator A is incremented in binary from 00, 01, 02…09, 0A, 0B,…, 0F, 10, 11,…FF

Note: To run this program, after selecting DEBUG session in the main menu use View-> Watch & call Stack window, in the Watches select watch 1(or 2) and press F2 and enter a (for accumulator A)

b. Write an assembly language program to implement (display) an eight bit UP/DOWN decimal counter on watch window.

mov a, #99h
back: acall delay
add a, #99h
daa
jnz back
here: sjmp here
delay: mov r1, #0ffh
decr1: mov r2, #0ffh
decr: mov r3, #0ffh
djnz r3, $
djnz r2, decr
djnz r1, decr1
ret
end

RESULT: Accumulator A is incremented in BCD from 99, 98, 97,……., 00.
Experiment no.5: Code Conversion Programming

a.  Write an assembly language program to convert a BCD number into ASCII.

```assembly
mov a, #09h       //the BCD number to be converted to ASCII
mov r0,a
swap a
mov dptr,#9000h  // output will be in 9000h and 90001h
acall ascii
mov a,r0
acall ascii
sjmp $

ascii: anl a,#0fh
add a,#30h
movx @dptr,a
inc dptr
ret
end
```

Result:

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

b. i  Write an assembly language program to convert a ASCII number into Decimal.

```assembly
mov dptr,#9000h  //ASCII no. to be converted to decimal is stored in 9000h
movx a,@dptr
subb a,#30h
inc dptr
movx @dptr,a
end
```

Result:

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

b. ii Write an assembly language program to convert a decimal number into ASCII.

```assembly
mov dptr,#9000h  //Decimal number to be converted to ASCII is store in 9000h
movx a,@dptr     // 9000h
add a,#30h
inc dptr
movx @dptr,a     // ASCII will be saved in 9001h
end
```

Result:

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
c. i Write an assembly language program to convert a binary (hex) number into decimal.

```
mov a,#0feh       //binary number to be converted to decimal
mov b,#0ah
div ab
mov r0,b
mov b,#0ah
div ab
mov 30h,a
mov a,b
swap a
orl a,r0
mov 31h,a
end
```

Result:

Input:  
Output:  

---

c. ii Write an assembly language program to convert a decimal number into binary(hex).

```
mov a,#95h          ///a = Decimal number to be converted to the binary
mov b,#10h
div ab
mov r1,b
mov b,#0ah
mul ab
add a,r1
mov 30h,a
end
```

Result:

Input:  
Output:  

---
Experiment no. 6: Serial Communication Programming

a. Conduct an experiment to configure 8051 microcontroller to transmit characters “ENTER YOUR NAME” to a PC using the serial port and display on the serial window.

**Note:** To use result of this program, after selecting DEBUG session in the main menu use View-> serial window #1. On running & halting the program, the data is seen in the serial window.

```asm
mov tmod,#20h  //setting Timer-1 in mode-2
mov scon,#70h
mov th1,#-3
setb tr1
again: mov r0,#03h
mov dptr,#8000h
nextchar: movx a,@dptr
         acall transfer
         inc dptr
         djnz r0,nextchar
         sjmp again
transfer: mov sbuf,a
wait:     jnb ti,wait
         clr ti
         ret
         end
```

**RESULT:**

Each time the program is executed, “ENTER YOUR NAME” will be displayed on the serial window.

**Baud rate Calculation:**

Crystal freq/ (12*32) = (11.0592MHz)/(12*32) = 28800.
Serial communication circuitry divides the machine cycle frequency (11.0592MHz)/(12) by 32 before it is being used by the timer to set the baud rate.

To get 9600, 28800/3 is obtained by loading timer1 with -3 (i.e., FF – 3 = FD) for further clock division. For 2400 baud rate, 28800/12 => -12 = F4 in TH1.
b. Conduct an experiment to generate 1 second delay continuously using on chip timer.

```
mov tmod,#02h
mov th0,#00h
clr P1.0
clr a
setb tr0
again: mov r7,#0ffh
loop: mov r6,#14d
wait: jnb tf0, wait
clr tf0
djnz r6,wait
djnz r7,loop
cpl P1.0
sjmp again
end
```

**RESULT:**

Accumulator A is incremented in binary from 00, 01, 02...09, 0A, 0B, ..., 0F, 10, 11, ...FF every 1 second (for 33MHz clock setting & every 3 seconds for 11.0598MHz)
Experiment no.7: Waveform generation using DAC

a. Write a C program to generate square wave of amplitude ___ V & frequency _______Hz using DAC. Display the waveform on the CRO.

Circuit Diagram for waveform generation:
**Program:**

```c
#include <REG51xD2.H>
void delay(unsigned int x)  /* delay routine */
{
    for(;x>0;x--);
}

main()
{
    unsigned char on = 0x7f,off=0x00;
    unsigned int fre = 230;

    while(1)
    {
        P0=P1=on; /* write amplitude to port */
        delay(fre);
        P0=P1=off; /* clear port */
        delay(fre);
    }
}
```

**DESIGN:**

Let \( f = 2 \) kHz, Therefore \( T = \frac{1}{f} = 0.5 \) msec.
Count value for the delay is \( \frac{T}{\text{1clock cycle period}} = \frac{0.5 \times 10^{-3}}{1.085 \times 10^{-6}} \) sec
Hence Count value is \( \frac{460}{230} \).
Hence for 50% Duty cycle the Count value is half of the Count value=230.

**Note:** Delay produced by the program will depend on the microcontroller you are using, so frequency of the waveform generated may not match with the given frequency.

**b. Write a C program to generate ramp wave of amplitude ____ V using DAC. Display the waveform on the CRO.**

**Program:**

```c
#include <REG51xD2.H>
main()
{
    unsigned char amp = 0xff;
    unsigned char i=0;
    P0=P1=0x00; /* P0 as Output port */
    while(1)
    {
        for(i=0;i<amp;i++) /* Generate ON pulse */
            P0=P1=i;
    }
}
```
c. Write a C program to generate triangular wave of amplitude ___ V using DAC. Display the waveform on the CRO.

Program:
```
#include <REG51xD2.H>
main()
{
    unsigned char i=0;
    P0=P1=0x00;    /* P0 as Output port */
    while(1)
    {
        for(i=0x00;i<0xff;i++) /* Generate ON pulse */
            P0=P1=i;

        for(i=0xff;i>0x00;i--)
            /* Generate OFF pulse */
            P0=P1=i;
    }
}
```

d. Program for dual DAC interfacing to generate sine waveform.

To generate a sine wave, we first need a table whose values represent the magnitude of the sine of angles between 0 360 degrees. The values for the sine function vary form -1.0 to +1.0 for 0- to 360- degree angles. Therefore, the table values are integer numbers representing the voltage magnitude for the sine of theta. This method ensures that only integer numbers are output to the DAC by the 8051 microcontroller. Table below shows the angles, the sine values, the voltage magnitudes, and the integer values representing the voltage magnitude for each angle. To generate table, we assumed the full-scale voltage of 10 V for DAC output. Full-scale output of the DAC is achieved when all the data inputs of the DAC are high. Therefore, to achieve the full-scale 10 V output, we use following equation.

\[ V_{out} = 5V + (5V \times \sin \theta) \]

Program:
```
#include <REG51xD2.H>
void main()
{
    unsigned char i, w;
    wave[36]={128,148,171,192,209,225,238,245,253,255,253,245,238,225,209,192,171,128,104,82,64,43,28,15,07,01,00,01,07,15,28,43,64,82,104};
    P0 = 0x00;
    while(1)
    {
        for (i=0; i<36; i++)
            P0=P1=wave[i];
    }
}``
<table>
<thead>
<tr>
<th>Angle ‘θ’ in degrees</th>
<th>Sinθ</th>
<th>$V_{out} = 5V + (5V \times \sinθ)$</th>
<th>Values sent to DAC (decimal) $V_{out} \times 25.6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>70</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>90</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>120</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>130</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>140</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>160</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>170</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>180</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>190</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>210</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>220</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>230</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>240</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>250</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>260</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>270</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>280</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>290</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>300</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>310</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>320</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>330</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>340</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>350</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>360</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: Difference in angles should be 10 degree.
Experiment no.8: Motor Interfacing

a. Write a C program to interface stepper motor.

Circuit Diagram:

```
8  P 0.0
0
5  P 0.7
1
```

Theory:

Stepper motor is an electromechanical device which converts electrical pulses into discrete mechanical movements. The shaft or spindle of a stepper motor rotates in discrete step increments when electrical command pulses are applied to it in the proper sequence. The motors rotation has several direct relationships to these applied input pulses. The sequence of the applied pulses is directly related to the direction of motor shafts rotation. The speed of the motor shafts rotation is directly related to the frequency of the input pulses and the length of rotation is directly related to the number of input pulses applied.

Stepper Motor Advantages:

1. The rotation angle of the motor is proportional to the input pulse.
2. The motor has full torque at standstill (if the windings are energized)
3. Precise positioning and repeatability of movement since good stepper motors have an accuracy of 3 – 5% of a step and this error is non cumulative from one step to the next.
4. Excellent response to starting/stopping/reversing.
5. Very reliable since there are no contact brushes in the motor. Therefore the life of the motor is simply dependent on the life of the bearing.
6. The motors response to digital input pulses provides open-loop control, making the motor simpler and less costly to control.
7. It is possible to achieve very low speed synchronous rotation with a load that is directly coupled to the shaft.
8. A wide range of rotational speeds can be realized as the speed is proportional to the frequency of the input pulses.

Stepper Motor Disadvantages:

1. Resonances can occur if not properly controlled.
2. Not easy to operate at extremely high speeds.

Open Loop Operation:

One of the most significant advantages of a stepper motor is its ability to be accurately controlled in an open loop system. Open loop control means no feedback information about position is needed. This type of control eliminates the need for expensive sensing and feedback devices such as optical encoders. Your position is known simply by keeping track of the input step pulses.
#include <REG51xD2.H>
void delay (unsigned int x) /* Delay Routine */
{
    for(;x>0;x--);
    return;
}

main ( )
{
    unsigned char Val, i;
    P0=0x00;
    while(1)
    {
        Val = 0x11;
        for (i=0;i<4;i++)
        {
            P0 = Val;
            Val = Val<<1; /* Val= Val>>1; for clockwise direction*/
            delay (500);
        }
    }
}

Motor Specifications:
    Step Angle = 1.8 degrees
    Step angle accuracy = 5%
    Holding Torque = 40Ncm
    Rotor Inertia = 115gcm²
    Weight = 0.5Kg
    Insulation = Class B
b. Write a C program to interface DC motor.

Circuit Diagram:

```
  8   P 3.2
 0   (INCR)
 5   P 3.3
 1   (DECR)
```

Theory:

DC motors are used in many applications like process control and automation in an industry, robotics, consumer electronics, office automation equipment like printers and scanners etc. One can consider the use of a DC motor wherever there is need to control the motion of an object. Speed control of the motor is important in the applications involving them. For example, in an audio system, the DC motor that drives the cassette should always run at a fixed speed. Like wise, there are applications where the speed of the DC motor has to change according to some defined conditions. The DC motor used in this interface module is a 12V, 4W motor that can be seen in many electronic equipments. The circuit to control the speed of the motor follows a general concept and can be applied to DC motors of higher capacity also.

The pulse width modulation technique is used to vary the speed of the DC motor. The frequency of the pulses is 120Hz. Keeping the frequency constant, the width of the pulses is used to change the speed. When the pulse width is minimum, the speed is minimum and when the width is maximum, the speed is maximum (2400rpm). The ramp and pedestal technique is used to change the pulse width and thereby the speed.
Program:

```c
#include <REG51xD2.H>
sbit inr = P3^2;  //speed increment switch
sbit dcr = P3^3;  //speed decrement switch

main()
{
    unsigned char i=0x80;
P0 = 0x7f;  /*Run the motor at half speed.*/
while (1)
{
    if (!inr)
    {
        while (!inr);
        if(i>10)
            i=i-10;  //increase the DC motor speed
    }
    if(!dcr)
    {
        while(!dcr);
        if(i<0xf0)
            i=i+10;  //decrease the DC motor speed
    }
P0=i;
}
```

Experiment no. 9: LCD and keypad interfacing

Block Diagram:

HEX values of the keys:

<table>
<thead>
<tr>
<th>LABEL ON THE KEYTOP</th>
<th>HEX CODE</th>
<th>LABEL ON THE KEYTOP</th>
<th>HEX CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>-</td>
<td>0C</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>*</td>
<td>0D</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>/</td>
<td>0E</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>%</td>
<td>0F</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>AC</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>CE</td>
<td>11</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>CHK</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>=</td>
<td>13</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>MC</td>
<td>14</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>MR</td>
<td>15</td>
</tr>
<tr>
<td>.</td>
<td>0A</td>
<td>M</td>
<td>16</td>
</tr>
<tr>
<td>+</td>
<td>0B</td>
<td>M+</td>
<td>17</td>
</tr>
</tbody>
</table>
Program to interface LCD and KEYPAD:

```c
#include <REG51xD2.H>
#include "lcd.h"
unsigned char getkey();
void delay(unsigned int);
main()
{
    unsigned char key,tmp;
    InitLcd(); //Initialise LCD
    WriteString("Key Pressed="); // Display msg on LCD
    while(1)
    {
        GotoXY(12,0); //Set Cursor Position
        key = getkey(); //Call Getkey method
    }
}

unsigned char getkey()
{
    unsigned char i,j,k,indx,t;
    P2 = 0x00; //P2 as Output port
    indx = 0x00; //Index for storing the 1st value of scanline
    for(i=1;i<=4;i<<=1) //for 3 scanlines
    {
        P2 = 0x0f & ~i; //write data to scanline
        t = P0; //Read readlines connected to P0
        t = ~t;
        if(t>0) //If key press is true
        {
            delay(6000); //Delay for bouncing
            for(j=0;j<=7;j++) //Check for 8 lines
            {
                t >>=1;
                if(t==0) //if get pressed key
                {
                    k = indx+j; //Display that by converting to Ascii
                    t = k>>4;
                    t +=0x30;
                    WriteChar(t); //Write upper nibble
                    t = k & 0x0f;
                    if(t > 9)
                        t+=0x37;
                    else
                        t+=0x30;
                    WriteChar(t); //write lower nibble
                    return(indx+j); //Return index of the key pressed
                }
            }
            indx += 8; //If no key pressed increment index
        }
    }
}
void delay(unsigned int x) //Delay routine
{
    for(;x>0;x--); }
```
Experiment no.10: C program to interface Elevator

Block Diagram:

Theory:
The operation of the elevator is as follows:

- Initially, the elevator is at ground floor.
- When the elevator reaches any floor, it stays at that floor until a request from another floor is made. When such a request is detected, it moves to that floor.
- The floor request are scanned in fixed order i.e., floors 0, 1, 2 and 3.

This interface simulates the control and operation of an elevator. Four floors assumed and for each floor a key and corresponding LED indicator are provided to serve as request buttons and request status indicator. The elevator itself is represented by a column of ten LEDs. The motion of elevator can be simulated by turning on successive LEDs one at a time. The delay between turning off one LED and turning on the next LED can simulate the “speed” of the elevator. User can read the request status information through one port, reset the request indicators through another port and control the elevator (LED column) through another port.

Description of the Circuit

This interface has four keys, marked 0, 1, 2, and 3 representing the request buttons at the four floors. Pressing of key causes a corresponding Flip-Flop to be set. The outputs of the four Flip-flops can be read through port B (PBO, PBI, PB2 and PB3). Also, the status of these signals is reflected by a set of 4 LEDs. The Flip-Flop can be rest (LEDs are cleared) through port A (PA54, PA5, PA6, and PA7). A column of 10 LEDs, representing the elevator can be controlled through Port A (PA0, PA1, PA2 and PA3). These port lines are fed to the inputs of the decoder 7442 whose outputs are used to control the on/off states of the LEDs which simulate the motion of the elevator.
#include <REG51xD2.H>
void delay(unsigned int);

main()
{
    unsigned char Flr[9]={0xff,0x00,0x03,0xff,0x06,0xff,0xff,0xff,0x09};
    unsigned char FClr[9]={0xff,0x0e0,0x0d3,0xff,0x0b6,0xff,0xff,0xff,0x79};
    unsigned char ReqFlr,CurFlr = 0x01,i,j;

    P0 = 0x00;
    P0 = 0x0f0;

    while(1)
    {
        P1 = 0x0f;
        ReqFlr = P1 | 0x0f0;
        while(ReqFlr == 0x0ff)
            ReqFlr = P1 | 0x0f0; //Read Request Floor from P1
        ReqFlr = ~ReqFlr;

        if(CurFlr == ReqFlr)  //If Request floor is equal to Current Floor
        {
            P0 = FClr[CurFlr]; //Clear Floor Indicator
            continue;            //Go up to read again
        }

        else if(CurFlr > ReqFlr)  //If Current floor is > request floor
        {
            i = Flr[CurFlr] - Flr[ReqFlr]; //Get the no of floors to travel
            j = Flr[CurFlr];
            for(;i>0;i--)                 // Move the indicator down
            {
                P0 = 0x0f0|j;
                j--;
                delay(50000);
            }
        }

        else  // If Current floor is < request floor
        {
            i = Flr[ReqFlr] - Flr[CurFlr]; //Get the no of floors to travel
            j = Flr[CurFlr];
            for(;i>0;i--)                 // Move the indicator Up
            {
                P0 = 0x0f0 | j;
                j++;
                delay(50000);
            }
        }

        CurFlr = ReqFlr;  // Update Current floor
P0 = FClr[CurFlr];  // Clear the indicator
}

void delay(unsigned int x)
{
    for(;x>0;x--);
}
Experiment no.11: Write a C program to interface Calculator using Keyboard and 7-segment display.

Algorithm:
- Read the numbers n1 and n2 from keyboard and display them on seven segment.
- Read the operand from the keypad if key pressed is B (+), C (-), D (*), E (/) then respective operation is performed.
- Result is displayed on 2 digit seven segment display.
- If any time the key pressed value returned as 10h then clear the LCD.

Program for calculator
#include <REG51xD2.H>
void DispChar(unsigned char ch);
void ClrLED();
unsigned char getkey();
unsigned char getnum();
unsigned char getOp();
sbit Clk = P3^4; /* Clock line for 7 segment display */
sbit Dat = P0^0; /* Data line for 7 segment display */
main()
{
  unsigned char tmp=0x0ff,n1=0,n2,Op,Res;
  unsigned char NumTab[10] =
  { 0x0c0,0x0f9,0x0a4,0xb0,0x99,0x92,0x82,0x0f8,0x80,0x90 };
  unsigned char OpTab[4] = { 0x88,0x0Bf,0xc8,0x0a1};
  bit Neg=0;
  ClrLED();        /* Clear 7 segment display */
  while(1)
  {
    Neg = 0;        /* Negative flag */
    n1=getnum();    /* Get 1st number */
    Op = getOp() - 0x0B; /* Get Opcode. 0x0b is keycode of '+'(see keyboard schematics)*/
  
}
n2=getnum(); /* Get 2nd number */
while(getkey()!=0x13); /* wait for '=' key */
ClrLED();
switch(Op) /* Perform corresponding operation */
{
    case 0: Res = n1 + n2; break;
    case 1:
        if(n2>n1) /* check for negativity */
        {
            Neg = 1;
            Res = n2 - n1;
            break; }
        Res = n1 - n2;
        break;
    case 2: Res = n1 * n2; break;
    case 3:
        Res = n1 / n2;
        break;
    }
DispChar(NumTab[Res%10]); /* Display number */
DispChar(NumTab[Res/10]);
if(Neg) /* if negative result display '-' */
DispChar(0x0Bf);
}
void DispChar(unsigned char ch) /* Routine to display char on 7 segment */
{
    unsigned char i,tmp;
    P0=0x00;
    for(i=0;i<8;i++) /* for all bits */
    {
        tmp = ch & 0x80;
        if(tmp)/* write data depending on MSB */
            Dat = 1;
        else
            Dat = 0;
        Clk = 0; /* Give Clk Pulse for synchronization */
        Clk = 1;
        ch = ch << 1; /* Get next bit */
    }
}
void ClrLED()
{
    unsigned char i;
    for(i=0;i<4;i++)
        DispChar(0x0ff); /* 0xff for clear segment ( see 7 segment manual for more info */
}
unsigned char getkey()
{
    unsigned char i,j,indx,t;

    P2 = 0x00;           /* P2 as Output port */
    P0 = 0x0ff;
    indx = 0x00;        /* Index for storing the first value of scanline */
    while(1)
    {
        for(i=1;i<=4;i<<=1)     /* for 4 scanlines */
        {
            P2 = 0x0f & ~i;      /* write data to scanline */
            t = P0;            /* Read readlines connected to P0*/
            t = ~t;
            if(t>0)              /* If key press is true */
            {
                for(j=0;j<=7;j++)    /* Check for 8 lines */
                {
                    t >>=1;
                    if(t==0)          /* if get pressed key*/
                    {
                        return(indx+j);  /* Return index of the key pressed */
                    }
                }
            }
            indx += 8;      /* If no key pressed increment index */
        }
    }

    unsigned char getnum()        /* Method for getting number */
    {
        unsigned char tmp;
        while(1)
        {
            tmp = getkey();
            if(tmp < 0x0a || tmp==0x10) /* if pressed key is number, return */
                return(tmp);
        }
    }

    unsigned char getOp()         /* Method for getting Operator */
    {
        unsigned char tmp;
        while(1)
        {
            tmp = getkey();
            if((tmp > 0x0a && tmp <0x0f)|| tmp==0x10) /* if pressed key is a Operator, return */
                return(tmp);
        }
    }
Experiment no.12: C program to interface Temperature Sensor
Program for temperature sensor.

```c
#include <at89c51xd2.h>
#include<intrins.h>
#include "lcd.h"

unsigned int Adc;
unsigned char Low_adc,High_adc,relay;

read_adc()
{
    unsigned char status;

    P2_3 = 1; // Start conversion of ADC
    status = P1; //Read status of ADC
    while((status & 0x01) != 0x01)
    {
        status = P1;
    }
    P2_2 = 0; // Enable outputs
    P2_0 = 0; // Activate B1 to B8 outputs
    Low_adc = P0; // Read lower byte of ADC and place in R0
    P2_0 = 1; // Deactivate B1 to B8 outputs
    P2_1 = 0; // Activate B9 to B12 and POL, over range outputs
    High_adc = P0; // Read higher byte of ADC
    High_adc = High_adc & 0x0F;
    P2_1 = 1; // deactivate B9 to B12 and
    POL, over range outputs
    P2_2 = 1; // Disable outputs
    P2_3 = 0; // Stop conversion of ADC
}

main()
{
    float Temp,Vol,Res;
    unsigned char Temp1;
    unsigned char Temp2,Temp3;
    P0 = 0xFF; // Make port 0 as input
    P2 = 0xFF; // Make port 2 as high now the relay is on.
    P1_1 = 0; // switch OFF relay
    P2_3 = 0; // STOP conversion of ADC
    relay = 10;

    while(1)
    {
        // Rest of the main function code...
    }
```
read_adc();  //Read ADC  
    Adc = High_adc;
    Adc <<= 8; 
    Adc = Adc | Low_adc;
    if( (Adc > 0x656) && (relay != 0))  //IF greater than 0x0656 Switch OFF relay
        {
            ClrLcd();
            WriteString("RELAY OFF");
            P1_1 = 0    ;
            relay = 0;
        }
    else if ( (Adc < 0x5b9) && (relay!= 1)) //IF less than 0x05B9 Switch ON relay
        {
            ClrLcd();
            WriteString("RELAY ON");
            P1_1 = 1    ;
            relay = 1;
        }

    Vol =-((Adc/10)*0.000488);       //voltage before amplifier

    Res =((100*(1.8-Vol)-100*Vol)*100)   /(100*Vol + 100*(1.8+Vol));//Resistance Value
    Res = Res - 100;
    Temp = Res/ 0.384;

    Temp1 = Temp;
    Temp2 = 0x30 + (Temp1 / 0x0A); 
    Temp3 = 0x30 + (Temp1 % 0x0A);

    GotoXY(0,1);
    WriteString("Temperature ");
    WriteChar(Temp2);
    WriteChar(Temp3);
    WriteString("'C");

    }
The features of the MSP430 Microcontroller are:

- Ultra-low power; mixed signal processors.
- Widely used in battery operated applications.
- Uses Von Neumann architecture to connect CPU, peripherals and buses.
- 1 to 60 kB flash
- 256B to 2kB RAM
- SMD package with 20 to 100 pins
- MSP 430 family has 4 kB flash, 256B RAM, 2 timers.

Architecture: Basic Elements

- 16 bit RISC processor
- Programmable 10/12 bit ADC
- 12 bit Dual DAC for accurate analog voltage representation
- Supply voltage supervisor for detection of Gray level
- Programmable timers, Main and Auxiliary crystal circuits

CPU features

- Reduced Instruction Set Computer Architecture
- 27 instructions wide instruction set
- 7 orthogonal addressing modes
- Memory to Memory data transfer
- Separate 16 bit Address and Data buses
- 2 constant number generators to optimize code
STEPS TO CREATE AND COMPILE MSP430 PROJECT IN IAR-EMBEDDED WORKBENCH:

12. Double Click on the IAR-embedded workbench icon.
13. Close any previous projects that were opened using – Project ➔ Close.

15. 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 right click on the Source Group in the Project Window and the Add Files to Group option.

16. Compile the project; using this button. Then click make. Any errors in the code are indicated by the window shown in figure below, along with the error line. Debug the errors. After an error free, go to Download and Debug mode by using this icon.
17. After clearing errors and clicking download & debug button below window will appear.

18. Run the program using these icons.

19. To stop debugging click this icon.
13. Data Transfer and Arithmetic Instruction Programming

a. Write an assembly language program to transfer $N = ___$ bytes of data from location $A:_______h$ to location $B:_______h$ (without overlap).

$A=0x8000$, $B=0x9000$, $N=5$

```assembly
#include "msp430.h" ; define controlled include file
NAME main ; module name
PUBLIC main ; make the main label visible outside this module
ORG 0xFFFFe
DC16 init ; set reset vector to 'init' label
RSEG CSTACK ; pre-declaration of segment
RSEG CODE ; place program in 'CODE' segment

init: MOV #SFE(CSTACK), SP ; set up stack
main: NOP ; main program
      MOV.W #WDTPW+WDTHOLD,&WDTCTL ; Stop watchdog timer

MOV.W #0x8000, R5
MOV.W #0x9000,R6
MOV.B #5,R7
again: MOV.W @R5+,0(R6)
       INCD.W R6
       DEC R7
       JNZ again
       JMP $

END```

Result:

Input:

<table>
<thead>
<tr>
<th>Go to</th>
<th>0x8000</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>8000</td>
<td>1111</td>
<td>2222</td>
</tr>
<tr>
<td>8010</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>8020</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>8030</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>8040</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>8050</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>8060</td>
<td>0000</td>
<td>0000</td>
</tr>
</tbody>
</table>

Output:

<table>
<thead>
<tr>
<th>Go to</th>
<th>0x9000</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>9000</td>
<td>1111</td>
<td>2222</td>
</tr>
<tr>
<td>9010</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>9020</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>9030</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>9040</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>9050</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>9060</td>
<td>0000</td>
<td>0000</td>
</tr>
</tbody>
</table>
b. Write an assembly language program to exchange \( N = \_\_h \) bytes of data at location \( A : \_\_\_h \) and at location \( B : \_\_\_h \).

\( A = 0x8000, B = 0x9000, N = 5 \)

```assembly
#include "msp430.h" ; #define controlled include file
NAME    main ; module name
PUBLIC  main ; make the main label visible outside this module
ORG     0FFFEh
DC16    init ; set reset vector to 'init' label
RSEG    CSTACK ; pre-declaration of segment
RSEG    CODE ; place program in 'CODE' segment
init:   MOV $FE(CSTACK), SP ; set up stack
main:   NOP ; main program
        MOV.W #WDTPW+WDTHOLD,&WDTCTL ; Stop watchdog timer

MOV.W #0x8000, R5
MOV.W #0x9000, R6
MOV.B #5, R7

again:  MOV.W @R5, R8
        MOV.W @R6, 0(R5)
        MOV.W R8, 0(R6)
        INCD.W R6
        INCD.W R5
        DEC R7
        JNZ again
        JMP $
END
```

Result:

Input: 

Output:
c. Write an assembly language program to sort an array of \( N = \_\_\_\_\_ \) h bytes of data in ascending/descending order stored from location 9000h. (use bubble sort algorithm)

```assembly
#include "msp430.h" ; #define controlled include file
NAME main ; module name
PUBLIC main ; make the main label visible outside this module
ORG 0FFFEh
DC16 init ; set reset vector to 'init' label
RSEG CSTACK ; pre-declaration of segment
RSEG CODE ; place program in 'CODE' segment
init: MOV #SFE(CSTACK), SP ; set up stack
main: NOP ; main program
MOV.W #WDTPW+WDTHOLD,&WDTCTL ; Stop watchdog timer
MOV.W #04,R4 ; count (N-1) ARRAY SIZE=N
UP:
    MOV.W #0x9000,R10 ; array stored from address 9000h
    MOV.W #00,R11
    MOV.W R4, R5 ; initialize exchange counter
REPEAT:
    MOV.W @R10+, R6 ; Get 1st Number from Array
    MOV.W R6, R8
    MOV.W @R10, R7 ; Get 2nd Number from Array
    MOV.W R7, R9
    SUB.W R7, R6
    JNC NOEXCHG ; jc - for descending order
    MOV.W R8, 0(R10) ; Exchange the 2 No's In the Array
    DEC.W R10
    DEC.W R10
    MOV.W R9, 0(R10)
    INCD.W R10
NOEXCHG:
    DEC.W R5
    CMP.W R11, R5
    JNE REPEAT
    DEC.W R4
    CMP.W R11, R4
    JNE UP
    JMP $ ; set up stack
    END
```

**Note:** For smallest number take the first element in the ascending order sorted array and for largest number take the first element in the descending order sorted array.
d. Write an assembly language program to perform the addition of two 32-bit numbers.

```assembly
#include "msp430.h" ; #define controlled include file
NAME main ; module name
PUBLIC main ; make the main label visible outside this module
ORG 0FFFEh ; set reset vector to 'init' label
RSEG CSTACK ; pre-declaration of segment
RSEG CODE ; place program in 'CODE' segment

init: MOV #SFE(CSTACK), SP ; set up stack
main: NOP ; main program

MOV.W #WDTPW+WDT HOLD,&WDTCTL ; Stop watchdog timer
MOV.W #0X9000,R4 //NUM1:FFFF9000
MOV.W #0xFFFF,R7 //NUM2:FFFFFFFF
ADD.W R4,R7
MOV.W #0xFFFF,R5
MOV.W #0xFFFF,R6
ADDC R5,R6
JMP $
END
```

Result:

Input: Output:
Write an assembly language program to perform the subtraction of two 32-bit numbers.

```assembly
#include "msp430.h" ; define controlled include file
NAME main ; module name
PUBLIC main ; make the main label visible outside this module
ORG 0FFFEh ; set reset vector to 'init' label
RSEG CSTACK ; pre-declaration of segment
RSEG CODE ; place program in 'CODE' segment

init:   MOV #SFE(CSTACK), SP ; set up stack
main:   NOP ; main program
        MOV.W #WDTPW+WDTHOLD,&WDTCTL ; Stop watchdog timer
        MOV.W #0X9000,R4
        MOV.W #0XFFFF,R7
        SUB.W R4,R7
        MOV.W #0XFFFF,R5
        MOV.W #0XFFFF,R6
        SUBC R5,R6
        JMP $
END
```

**Result:**

<table>
<thead>
<tr>
<th>Input:</th>
<th>Output:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
f. Write an assembly language program to perform the multiplication of two 16-bit numbers.

```assembly
#include "msp430.h"            ; #define controlled include file
NAME    main                    ; module name
PUBLIC  main
ORG     0FFEh
DC16    init                   ; set reset vector to 'init' label
RSEG    CSTACK                  ; pre-declaration of segment
RSEG    CODE

init:   MOV     #SFE(CSTACK), SP        ; set up stack
main:   NOP                             ; main program
        MOV.W   #WDTPW+WDTHOLD,&WDTCTL  ; Stop watchdog timer
        MOV.W #0XFFFF, R4             ; R4= FFFF
        MOV.W R4, R8
        MOV.W #0X1234, R7            ; R7= 1234, (FFFF x 1234)
        MOV.W #00, R5
        MOV.W #00, R10                ; PRODUCT LOWER 16 BIT (DB98)
        MOV.W #00, R9
CLRC
        INC.W R5
UP:     ADD.W R4, R8             ; SUCCESSIVE ADDITION
JNC COPY
        INC.W R9
COPY:   INC.W R5
CLRC
        CMP.W R5, R7
JNE UP
        MOV.W R8, R10
JMP $
END

Result:

Input:          Output:

Note: For square of a number give both the numbers same value.

Assignment: Find the square and cube of a number.
14. Boolean & Logical Instruction Programming

a. Write an assembly language program to count number of ones and zeros in an 8 bit number.

```assembly
#include "msp430.h" ; #define controlled include file

NAME   main ; module name
PUBLIC main ; make the main label visible outside this module

ORG 0FFFFh
DC16 init ; set reset vector to 'init' label
RSEG CSTACK ; pre-declaration of segment
RSEG CODE ; place program in 'CODE' segment

init: MOV #SFE(CSTACK), SP ; set up stack
main: NOP ; main program

MOV.W #WDTPW+WDTHOLD,&WDTCTL ; Stop watchdog timer
MOV.W #0000h,R4
MOV.W #0000h,R5
MOV.W #0008h,R6
MOV.W #0097h,R7

AGAIN: RRC R7
JC NEXT
INC R4
JMP HERE

NEXT: INC R5
HERE: DEC R6

JNZ AGAIN
JMP $

END

Results:
```

```
```
b. Write an assembly language program to find whether the given 8 bit number is even or odd.

```assembly
#include "msp430.h" ; #define controlled include file
NAME main ; module name
PUBLIC main ; make the main label visible outside this module
ORG 0FFFEh
DC16 init ; set reset vector to 'init' label
RSEG CSTACK ; pre-declaration of segment
RSEG CODE ; place program in 'CODE' segment
init: MOV #SFE(CSTACK), SP ; set up stack
main: NOP ; main program
MOV.W #WDTPW+WDTHOLD,&WDTCTL ; Stop watchdog timer
```

Results:

C. Write an assembly language program to perform logical operations AND, OR, XOR on two 16 bit numbers.

```assembly
#include "msp430.h" ; #define controlled include file
NAME main ; module name
PUBLIC main ; make the main label visible outside this module
ORG 0FFFEh
DC16 init ; set reset vector to 'init' label
RSEG CSTACK ; pre-declaration of segment
RSEG CODE ; place program in 'CODE' segment
init: MOV #SFE(CSTACK), SP ; set up stack
main: NOP ; main program
MOV.W #WDTPW+WDTHOLD,&WDTCTL ; Stop watchdog timer
MOV.W #0X1234, R5
MOV.W #0XABCD,R6
MOV.W R6,R7
MOV.W R6,R8
AND.W R5,R6 //R6=R5 AND R6
XOR.W R5,R7 //R7=R5 XOR R7
INV.W R8 //R8=NOT R8
INV.W R5
AND.W R8,R5
INV.W R5 //R5=R8 OR R5
JMP $
```

Results:
15. Counter and Code Conversion Programming

a. Write an assembly language program to implement (display) an 16 bit UP/DOWN binary (hex).

```assembly
#include "msp430.h" ; #define controlled include file
NAME main ; module name
PUBLIC main ; make the main label visible outside this module
ORG 0FFFEh
DC16 init ; set reset vector to 'init' label
RSEG CSTACK ; pre-declaration of segment
RSEG CODE ; place program in 'CODE' segment
init: MOV #SFE(CSTACK), SP ; set up stack
main: NOP ; main program
       MOV.W #WDTPW+WDTHOLD,&WDTCTL ; Stop watchdog timer
AGAIN: MOV.W #0X0000,R5 //For DOWN Counter, MOV.W #0XFFFF, R5
       REP: CALL #DELAY
       ADD.W #0X0001,R5 //For DOWN counter, ADD.W #0XFFFF,R5
       JNZ REP
       JMP AGAIN
       JMP $

DELAY:
       MOV.W #0X50,R6
LOOP1: MOV.W #0X50,R7
LOOP:  DEC R7
       JNZ LOOP
       DEC R6
       JNZ LOOP1
       RET
       END

RESULT: R5 is incremented in binary from 0000, 0001, 0002…0009, 000A, 000B,…,000F,0010,0011,…FFFF,0000,0001, ……
b. Write an assembly language program to implement (display) an 16 bit UP/DOWN Decimal counter.

```plaintext
#include "msp430.h"            ; #define controlled include file
NAME main                    ; module name
PUBLIC main                   ; make the main label visible outside this module
ORG 0FFFEh
DC16 init                     ; set reset vector to 'init' label
RSEG CSTACK                    ; pre-declaration of segment
RSEG CODE                      ; place program in 'CODE' segment
init: MOV #SFE(CSTACK), SP     ; set up stack
main: NOP
      MOV.W #WDTPW+WDTHOLD,&WDTCTL ; Stop watchdog timer

AGAIN: MOV.W #0X9999,R5         //For UP Counter, MOV.W #0X00, R5
REP: CALL #DELAY
    CLRC
    DADD.W #0X9999,R5           //For UP counter, DADD.W #0X0001,R5
    JNZ REP
    JMP AGAIN
    JMP $                        

DELAY: MOV.W #0X50,R6
LOOP1: MOV.W #0X50,R7
LOOP: DEC R7
    JNZ LOOP
    DEC R6
    JNZ LOOP1
    RET
END

RESULT:
R5 is decremented in BCD from 9999, 9998, ……., 0000, 9999, 9998…….
c. Write an assembly language program to convert a 8-bit BCD number into ASCII.

```assembly
#include "msp430.h" ; #define controlled include file
NAME main ; module name
PUBLIC main ; make the main label visible outside this module
ORG 0FFFEh
DC16 init ; set reset vector to 'init' label
RSEG CSTACK ; pre-declaration of segment
RSEG CODE ; place program in 'CODE' segment
init: MOV #SFE(CSTACK), SP ; set up stack
main: NOP ; main program
        MOV.W #WDTPW+WDTHOLD,&WDTCTL ; Stop watchdog timer

MOV.B #0X12, R5
MOV.B R5,R6
AND.B #0X0F,R6
ADD.B #0X30,R6
AND.B #0XF0,R5
RRA.B R5
RRA.B R5
RRA.B R5
RRA.B R5
ADD.B #0X30,R5
MOV.B R5,R7
JMP $

END

Result:
Input: Output:
```
d. Write an assembly language program to convert a ASCII number into Decimal.

```assembly
#include "msp430.h" ; #define controlled include file
NAME main ; module name
PUBLIC main ; make the main label visible outside this module
ORG 0FFFEh
DC16 init ; set reset vector to 'init' label
RSEG CSTACK ; pre-declaration of segment
RSEG CODE ; place program in 'CODE' segment
init: MOV #SFE(CSTACK), SP ; set up stack
main: NOP ; main program
MOV.W #WDTPW+WDTHOLD,&WDTCTL ; Stop watchdog timer

MOV.B #0X35, R5
SUB.B #0X30,R5
MOV.B R5,R6
JMP $
END
```

Result:

Input: 

Output: 

e. Write an assembly language program to convert a Decimal number into ASCII.

```assembly
#include "msp430.h" ; #define controlled include file
NAME main ; module name
PUBLIC main ; make the main label visible outside this module
ORG 0FFFEh
DC16 init ; set reset vector to 'init' label
RSEG CSTACK ; pre-declaration of segment
RSEG CODE ; place program in 'CODE' segment
init: MOV #SFE(CSTACK), SP ; set up stack
main: NOP ; main program
MOV.W #WDTPW+WDTHOLD,&WDTCTL ; Stop watchdog timer

MOV.B #0X35, R5
ADD.B #0X30,R5
MOV.B R5,R6
JMP $
END
```

Result:

Input: 

Output:
f. Write an assembly language program to convert a binary (hex) number into decimal.

```assembly
#include "msp430.h" ; #define controlled include file
NAME main ; module name
PUBLIC main ; make the main label visible outside this module
ORG 0FFF
Eh
DC16 init ; set reset vector to 'init' label
RSEG CSTACK ; pre-declaration of segment
RSEG CODE ; place program in 'CODE' segment
init: MOV #SFE(CSTACK), SP ; set up stack
main: NOP ; main program
MOV.W #WDTPW+WDTHOLD,&WDTCTL ; Stop watchdog timer

MOV.B #0XFE,R5
MOV.B #0X0A,R6
CALL #AA
MOV.B R5,R9
MOV.B R7,R5
CALL #AA
AND.W #0X00FF,R7
SWPB R7
RLA.B R5
RLA.B R5
RLA.B R5
RLA.B R5
ADD.W R5,R7
ADD.W R9,R7
JMP $

AA:

MOV.B #0XFF,R7

LOOP:

INC R7
SUB.B R6,R5
JC LOOP
ADD.W #0x0A,R5
RET

END

Result:

Input: Output:
g. Write an assembly language program to convert a decimal number into binary(hex).

```assembly
#include "msp430.h" ; define controlled include file
NAME main ; module name
PUBLIC main ; make the main label visible outside this module
ORG 0FFFEh
DC16 init ; set reset vector to 'init' label
RSEG STACK ; pre-declaration of segment
RSEG CODE ; place program in 'CODE' segment
init: MOV $FE(CSTACK), SP ; set up stack
main: NOP ; main program
MOV.W #WDTPW+WDTHOLD,&WDTCTL ; Stop watchdog timer

MOV.B #0X99,R5
MOV.B #0X10,R6
MOV.B #0XFF,R7

LOOP: INC R7
SUB.B R6,R5
JC LOOP
ADD.B #0x10,R5
AND.W #0X00FF,R7
MOV.B #0X00,R8

AGAIN:ADD.B #0X0A,R8
DEC.R7
JNZ AGAIN
ADD.B R5,R8
JMP$

END
```

Result:

Input: Output:
Additional Programs

1. **Program to generate arithmetic progression.** \( T_n = a + (n-1)d \)

   ```
   clr c
   mov r1,#03h  //d
   mov r2,#01h  //a
   mov r3,#08h
   mov r0,#30h
   mov 30h,r2
   mov r4,#01h
   next:  mov a,r1
           mov b,r4
           mul ab
           addc a,r2
           inc r0
           mov @r0,a
           inc r4
           djnz r3,next
   end
   ```

2. **Program to find the 2 out of 5 code.**

   ```
   mov r0,#30h  ; r0 as pointer
   mov a,@r0    ; no. to the Acc.
   ANL a,#0e0h  ; checks the 1st 3 bits
   jnz notvalid ; checks the 1st 3 bits
   mov a,@r0
   mov r2,#5    ; r2 as counter
   mov r1,#00h  ; r0 stores no. of ones
   clr c
   check:
   rrc a
   jnc skip
   inc r1       ; if carry increment r1
   skip:
   djnz r2,check
   mov A,r1
   clr c
   subb A,#02h  ; (a)!= 0 ===> (r2)!= 2 no. is not valid.
   jnz notvalid ; (a)!= 0 ====> (r2)!= 2 no. is not valid.
   inc r0
   mov @r0,#0ffh ; valid 2 out of 5 code
   sjmp exit
   notvalid:
   inc r0
   mov @r0,#11h  ; in valid 2 out of 5 code
   exit:
   sjmp exit
   end
   ```
3. **Program to generate first ten Fibonacci numbers.**

   Mov dptr, #9000h
   Mov r3, #08h
   Movx a, @dptr
   Mov r0,a
   Inc dptr
   Movx a, @dptr

   Back: xch a, r0
   Add a,r0
   Inc dptr
   Movx @dptr,a
   Djnz r3,back
   Lcall 0003h

4. **Program to add multibyte numbers.**

   Mov dptr,#9000h
   Mov r1,#04h
   Mov r2,#90h
   Mov r3,#91h
   Mov r4,#92h
   Clr c
   Mov dph,r2
   Back: movx a, @dptr
   Mov r5,a
   Mov dph,r3
   Movx a,@dptr
   Addc a,r5

   //Note:For multibyte subtraction put subb a,r5
   Mov dph,r4
   Movx @dptr,a
   Inc dptr
   Djnz r1,back
   Jnc end1
   Mov a,#01h
   Movx @dptr, a
   End1:lcall 0003h
   End
REFERENCES


Viva Questions

1. What do you mean by Embedded System? Give examples.
2. Why are embedded Systems useful?
3. What are the segments of Embedded System?
4. What is Embedded Controller?
5. What is Microcontroller?
6. List out the differences between Microcontroller and Microprocessor.
7. How are Microcontrollers more suitable than Microprocessor for Real Time Applications?
8. What are the General Features of Microcontroller?
9. Explain briefly the classification of Microcontroller.
10. Explain briefly the Embedded Tools.
11. Explain the general features of 8051 Microcontroller.
12. How many pin the 8051 has?
13. Differentiate between Program Memory and Data Memory.
14. What is the size of the Program and Data memory?
15. Write a note on internal RAM. What is the necessity of register banks? Explain.
16. How many address lines are required to address 4K of memory? Show the necessary calculations.
17. What is the function of accumulator?
18. What are SFR’s? Explain briefly.
19. What is the program counter? What is its use?
20. What is the size of the PC?
21. What is a stack pointer (SP)?
22. What is the size of SP?
23. What is the PSW? And briefly describe the function of its fields.
24. What is the difference between PC and DPTR?
25. What is the difference between PC and SP?
26. What is ALE? Explain the functions of the ALE in 8051.
27. Describe the 8051 oscillator and clock.
28. What are the disadvantages of the ceramic resonator?
29. What is the function of the capacitors in the oscillator circuit?
30. Show with an example, how the time taken to execute an instruction can be calculated.
31. What is the Data Pointer register? What is its use in the 8051?
32. Explain how the 8051 implement the Harvard Architecture?
33. Explain briefly the difference between the Von Neumann and the Harvard Architecture.
34. Describe in detail how the register banks are organized.
35. What are the bit addressable registers and what is the need?
36. What is the need for the general purpose RAM area?
37. Write a note on the Stack and the Stack Pointer.
38. Why should the stack be placed high in internal RAM?
39. Explain briefly how internal and external ROM gets accessed.
40. What are the different addressing modes supported by 8051 Microcontroller?
41. Explain the Immediate Addressing Mode.
42. Explain the Register Addressing Mode.
43. Explain the Direct Addressing Mode.
44. Explain the Indirect Addressing Mode.
45. Explain the Code Addressing Mode.
46. Explain in detail the Functional Classification of 8051 Instruction set
47. What are the instructions used to operate stack?
48. What are Accumulator specific transfer instructions?
49. What is the difference between INC and ADD instructions?
50. What is the difference between DEC and SUBB instructions?
51. What is the use of OV flag in MUL and DIV instructions?
52. What are single and two operand instructions?
53. Explain Unconditional and Conditional JMP and CALL instructions.
54. Explain the different types of RETURN instructions.
55. What is a software delay?
56. What are the factors to be considered while deciding a software delay?
57. What is a Machine cycle?
58. What is a State?
59. Explain the need for Hardware Timers and Counters?
60. Give a brief introduction on Timers/Counter.
61. What is the difference between Timer and Counter operation?
62. How many Timers are there in 8051?
63. What are the three functions of Timers?
64. What are the different modes of operation of timer/counter?
65. Give a brief introduction on the various Modes.
66. What is the count rate of timer operation?
67. What is the difference between mode 0 and mode 1?
68. What is the difference Modes 0,1,2 and 3?
69. How do you differentiate between Timers and Counters?
70. Explain the function of the TMOD register and its various fields?
71. How do you control the timer/counter operation?
72. What is the function of TF0/TF1 bit
73. Explain the function of the TCON register and its various fields?
74. Explain how the Timer/Counter Interrupts work.
75. Explain how the 8051 counts using Timers and Counters.
76. Explain Counting operation in detail in the 8051.
77. Explain why there is limit to the maximum external frequency that can be counted.
78. What’s the benefit of the auto-reload mode?
79. Write a short note on Serial and Parallel communication and highlight their advantages and disadvantages.
80. Explain Synchronous Serial Data Communication.
81. Explain Asynchronous Serial Data Communication.
82. Explain Simplex data transmission with examples.
83. Explain Half Duplex data transmission with examples.
84. Explain Full Duplex data transmission with examples.
85. What is Baud rate?
86. What is a Modem?
87. What are the various registers and pins in the 8051 required for Serial communication? Explain briefly.
88. Explain SCON register and the various fields.
89. Explain serial communication in general (synchronous and asynchronous). Also explain the use of the parity bit.
90. Explain the function of the PCON register during serial data communication.
91. How the Serial data interrupts are generated?
92. How is data transmitted serially in the 8051? Explain briefly.
93. How is data received serially in the 8051? Explain briefly.
94. What are the various modes of Serial Data Transmission? Explain each mode briefly.
95. Explain with a timing diagram the shift register mode in the 8051.
96. What is the use of the serial communication mode 0 in the 8051?
97. Explain in detail the Serial Data Mode 1 in the 8051.
98. Explain how the Baud rate is calculated for the Serial Data Mode 1.
99. How is the Baud rate for the Multiprocessor communication Mode calculated?
100. Explain in detail the Multiprocessor communication Mode in the 8051.
101. Explain the significance of the 9th bit in the Multiprocessor communication Mode.
102. Explain the Serial data mode 3 in the 8051.
103. What are interrupts and how are they useful in Real Time Programming?
104. Briefly describe the Interrupt structure in the 8051.
105. Explain about vectored and non-vectored interrupts in general.
106. What are the five interrupts provided in the 8051?
107. What are the three registers that control and operate the interrupts in 8051?
108. Describe the Interrupt Enable (IE) special function register and its various bits.
109. Describe the Interrupt Priority (IP) special function register and its need.
110. Explain in detail how the Timer Flag interrupts are generated.
111. Explain in detail how the Serial Flag interrupt is generated.
112. Explain in detail how the External Flag interrupts are generated.
113. What happens when a high logic is applied on the Reset pin?
114. Why the Reset interrupt is called a non-maskable interrupt?
115. Why do we require a reset pin?
116. How can you enable/disable some or all the interrupts?
117. Explain how interrupt priorities are set? And how interrupts that occur simultaneously are handled.
118. What Events can trigger interrupts, and where do they go after getting triggered?
119. What are the actions taken when an Interrupt Occurs?
120. What are Software generated interrupts and how are they generated?
121. What is RS232 and MAX232?
122. What is the function of RS and E pins in an LCD?
123. What is the use of R/W pin in an LCD?
124. What is the significance of DA instruction?
125. What is packed and unpacked BCD?
126. What is the difference between CY and OV flag?
127. When will the OV flag be set?
128. What is an ASCII code?
Microcontrollers Lab-10ESL47

MICROCONTROLLER - LAB QUESTION BANK

1. a) Write an ALP to move a Block of N-data starting at location X to location Y using 8051/MSP430
b) Write a C program to interface stepper motor to 8051.

2. a) Write an ALP to find cube of given 8-bit data using 8051/MSP430.
b) Write a C program to interface stepper motor to 8051.

3. a) Write an ALP to implement a binary/decimal up/down counter using 8051/MSP430.
b) Write a C program to interface stepper motor to 8051.

4. a) Write an ALP to find the largest/smallest element in an array using 8051.
b) Write a C program to interface stepper motor to 8051.

5. a) Write an ALP to exchange two blocks of data present at location X and Y respectively using 8051/MSP430.
b) Write a C program to generate Sine waveform using DAC. Display the waveform on CRO.

6. a) Write an ALP to arrange a set of N 8-bit numbers starting at location X in ascending/descending order using 8051/MSP430.
b) Write a C program to generate triangular wave of amp = ____ (1V-5V) using DAC. Display the waveform on CRO.

7. a) Write an ALP to perform 16-bit multiplication using 8051/MSP430.
b) Write a C program to generate Ramp wave of amp = ____ (1V-5V) using DAC. Display the waveform on CRO.

8. a) Write an ALP to convert two digit BCD number to its equivalent ASCII value using 8051/MSP430.
b) Write a C program to generate square wave of amp = ____ (1V-5V) using DAC. Display the waveform on CRO.

9. a) Write an ALP to find whether the given number is palindrome or not using 8051.
b) Write a C program to generate Sine waveform using DAC. Display the waveform on CRO.
10. a) Write an ALP to convert given Hexadecimal number to its equivalent Decimal number using 8051/MSP430.
b) Write a C program to interface DC motor to 8051.

11. a) Write an ALP to convert given Decimal number to its equivalent Hexadecimal using 8051/MSP430.
b) Write a C program to interface DC motor to 8051.

12. a) Write an ALP to perform 16-bit addition/subtraction using 8051/MSP430.
b) Write a C program to interface DC motor to 8051.

13. a) Write an ALP to count number of 1’s and 0’s in the given 8-bit data using 8051/MSP430.
b) Write a C program to interface Elevator to 8051.

14. a) Write an ALP to convert given ASCII number to its equivalent Decimal number.
b) Write a C program to interface Elevator to 8051

15. a) Write an ALP to find whether given number is even or odd using 8051/MSP430.
b) Write a C program to interface LCD panel and Hex keypad to 8051.

16. a) Write an ALP to convert given Decimal number to its equivalent ASCII using 8051/MSP430.
b) Write a C program to interface LCD panel and Hex keypad to 8051.
# Annexure-A

## MCS-51 Instruction set

### Instruction Set

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Byte</th>
<th>Oscillator Period</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ARITHMETIC OPERATIONS (Continued)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC DPTR</td>
<td>Increment Data Pointer</td>
<td>1</td>
<td>24</td>
</tr>
<tr>
<td>MUL AB</td>
<td>Multiply A &amp; B</td>
<td>1</td>
<td>48</td>
</tr>
<tr>
<td>DIV AB</td>
<td>Divide A by B</td>
<td>1</td>
<td>48</td>
</tr>
<tr>
<td>DA A</td>
<td>Decimal Adjust</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td><strong>LOGICAL OPERATIONS</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ANL A,Rn</td>
<td>AND Register to Accumulator</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>ANL A, direct</td>
<td>AND direct byte to Accumulator</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>ANL A, @ Ri</td>
<td>AND indirect RAM to Accumulator</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>ANL A, # data</td>
<td>AND immediate data to Accumulator</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>ANL direct, A</td>
<td>AND Accumulator to direct byte</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>ORL A,Rn</td>
<td>OR register to Accumulator</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>ORL A, direct</td>
<td>OR direct byte to Accumulator</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>ORL A, @ Ri</td>
<td>OR indirect RAM to Accumulator</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>ORL A, # data</td>
<td>OR immediate data to Accumulator</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>ORL direct, A</td>
<td>OR Accumulator to direct byte</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>ORL direct, # data</td>
<td>OR immediate data to direct byte</td>
<td>3</td>
<td>24</td>
</tr>
<tr>
<td>XRL A,Rn</td>
<td>Exclusive-OR register to Accumulator</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>XRL A, direct</td>
<td>Exclusive-OR direct byte to Accumulator</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>XRL A, @ Ri</td>
<td>Exclusive-OR indirect RAM to Accumulator</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>XRL A, # data</td>
<td>Exclusive-OR immediate data to Accumulator</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>XRL direct, A</td>
<td>Exclusive-OR Accumulator to direct byte</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>XRL direct, # data</td>
<td>Exclusive-OR immediate data to direct byte</td>
<td>3</td>
<td>24</td>
</tr>
<tr>
<td>CLR A</td>
<td>Clear Accumulator</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>CPL A</td>
<td>Complement Accumulator</td>
<td>1</td>
<td>12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Byte</th>
<th>Oscillator Period</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LOGICAL OPERATIONS (Continued)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RL A</td>
<td>Rotate Accumulator Left</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>RLC A</td>
<td>Rotate Accumulator Left through the Carry</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>RR A</td>
<td>Rotate Right</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>RRC A</td>
<td>Rotate Accumulator Right through the Carry</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>SWAP A</td>
<td>Swap nibbles within the Accumulator</td>
<td>1</td>
<td>12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Byte</th>
<th>Oscillator Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV A,Rn</td>
<td>Move register to Accumulator</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>MOV A, direct</td>
<td>Move direct byte to Accumulator</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>MOV A, @ Ri</td>
<td>Move indirect RAM to Accumulator</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>MOV A, # data</td>
<td>Move immediate data to Accumulator</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>MOV Rn, A</td>
<td>Move Accumulator to register</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>MOV Rn, direct</td>
<td>Move direct byte to register</td>
<td>2</td>
<td>24</td>
</tr>
<tr>
<td>MOV Rn, # data</td>
<td>Move immediate data to register</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>MOV direct, A</td>
<td>Move Accumulator to direct byte</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>MOV direct, Rn</td>
<td>Move register to direct byte</td>
<td>2</td>
<td>24</td>
</tr>
<tr>
<td>MOV direct, direct</td>
<td>Move direct byte to direct byte</td>
<td>3</td>
<td>24</td>
</tr>
<tr>
<td>MOV direct, @ Ri</td>
<td>Move indirect RAM to direct byte</td>
<td>2</td>
<td>24</td>
</tr>
<tr>
<td>MOV direct, # data</td>
<td>Move immediate data to direct byte</td>
<td>3</td>
<td>24</td>
</tr>
<tr>
<td>MOV @ Ri, A</td>
<td>Move Accumulator to indirect RAM</td>
<td>1</td>
<td>12</td>
</tr>
</tbody>
</table>

All mnemonics copyrighted © Intel Corporation 1980
<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Byte</th>
<th>Oscillator Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA TRANSFER</td>
<td>(Continued)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOV @Ri, direct</td>
<td>Move direct byte to indirect RAM</td>
<td>2</td>
<td>24</td>
</tr>
<tr>
<td>MOV @Ri,#data</td>
<td>Move immediate data to indirect RAM</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>MOV DPTR,#data16</td>
<td>Load Data Pointer with a 16-bit constant</td>
<td>3</td>
<td>24</td>
</tr>
<tr>
<td>MOVCA, A+DPTR</td>
<td>Move Code byte relative to DPTR to Acc</td>
<td>1</td>
<td>24</td>
</tr>
<tr>
<td>MOVCA, A+PC</td>
<td>Move Code byte relative to PC to Acc</td>
<td>1</td>
<td>24</td>
</tr>
<tr>
<td>MOVX A,@rI</td>
<td>Move External RAM (8-bit addr) to Acc</td>
<td>1</td>
<td>24</td>
</tr>
<tr>
<td>MOVX A,DPTR</td>
<td>Move External RAM (16-bit addr) to Acc</td>
<td>1</td>
<td>24</td>
</tr>
<tr>
<td>MOVX @Ri,A</td>
<td>Move Acc to External RAM (8-bit addr)</td>
<td>1</td>
<td>24</td>
</tr>
<tr>
<td>MOVX @DPTR,A</td>
<td>Move Acc to External RAM (16-bit addr)</td>
<td>1</td>
<td>24</td>
</tr>
<tr>
<td>PUSH direct</td>
<td>Push direct byte onto stack</td>
<td>2</td>
<td>24</td>
</tr>
<tr>
<td>POP direct</td>
<td>Pop direct byte from stack</td>
<td>2</td>
<td>24</td>
</tr>
<tr>
<td>XCH A,Rn</td>
<td>Exchange register with Accumulator</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>XCH A, direct</td>
<td>Exchange direct byte with Accumulator</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>XCH A,@Ri</td>
<td>Exchange indirect RAM with Accumulator</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>XCHD A,@Ri</td>
<td>Exchange low-order Digit indirect RAM with Accumulator</td>
<td>1</td>
<td>12</td>
</tr>
</tbody>
</table>

**BOOLEAN VARIABLE MANIPULATION**

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Byte</th>
<th>Oscillator Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLR</td>
<td>Clear Carry</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>CLR</td>
<td>Clear direct bit</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>SETB</td>
<td>Set Carry</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>SETB</td>
<td>Set direct bit</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>CPL</td>
<td>Complement Carry</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>CPL</td>
<td>Complement direct bit</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>ANL</td>
<td>AND direct bit to CARRY</td>
<td>2</td>
<td>24</td>
</tr>
<tr>
<td>ANL</td>
<td>AND complement of direct bit to Carry</td>
<td>2</td>
<td>24</td>
</tr>
<tr>
<td>ORL</td>
<td>OR direct bit to Carry</td>
<td>2</td>
<td>24</td>
</tr>
<tr>
<td>ORL</td>
<td>OR complement of direct bit to Carry</td>
<td>2</td>
<td>24</td>
</tr>
<tr>
<td>MOV</td>
<td>Move direct bit to Carry</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>MOV</td>
<td>Move Carry to direct bit</td>
<td>2</td>
<td>24</td>
</tr>
<tr>
<td>JC</td>
<td>Jump if Carry is set</td>
<td>2</td>
<td>24</td>
</tr>
<tr>
<td>JNC</td>
<td>Jump if Carry not set</td>
<td>2</td>
<td>24</td>
</tr>
<tr>
<td>JB</td>
<td>Jump if direct Bit is set</td>
<td>3</td>
<td>24</td>
</tr>
<tr>
<td>JNB</td>
<td>Jump if direct Bit is Not set</td>
<td>3</td>
<td>24</td>
</tr>
<tr>
<td>JBC</td>
<td>Jump if direct Bit is set &amp; clear bit</td>
<td>3</td>
<td>24</td>
</tr>
</tbody>
</table>

**PROGRAM BRANCHING**

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Byte</th>
<th>Oscillator Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACALL</td>
<td>Absolute Subroutine Call</td>
<td>2</td>
<td>24</td>
</tr>
<tr>
<td>LCALL</td>
<td>Long Subroutine Call</td>
<td>3</td>
<td>24</td>
</tr>
<tr>
<td>RET</td>
<td>Return from Subroutine</td>
<td>1</td>
<td>24</td>
</tr>
<tr>
<td>RETI</td>
<td>Return from interrupt</td>
<td>1</td>
<td>24</td>
</tr>
<tr>
<td>AJMP</td>
<td>Absolute Jump</td>
<td>2</td>
<td>24</td>
</tr>
<tr>
<td>LJMP</td>
<td>Long Jump</td>
<td>3</td>
<td>24</td>
</tr>
<tr>
<td>SJMP</td>
<td>Short Jump (relative addr)</td>
<td>2</td>
<td>24</td>
</tr>
</tbody>
</table>

All mnemonics copyrighted ©Intel Corporation 1980
### Microcontrollers Lab-10ESL47

#### PROGRAM BRANCHING (Continued)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Byte</th>
<th>Oscillator Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMP</td>
<td>Jump indirect relative to the D PTR</td>
<td>1</td>
<td>24</td>
</tr>
<tr>
<td>JZ rel</td>
<td>Jump if Accumulator is Zero</td>
<td>2</td>
<td>24</td>
</tr>
<tr>
<td>JNZ rel</td>
<td>Jump if Accumulator is Not Zero</td>
<td>2</td>
<td>24</td>
</tr>
<tr>
<td>CJNE A, direct, rel</td>
<td>Compare direct byte to Acc and Jump if Not Equal</td>
<td>3</td>
<td>24</td>
</tr>
<tr>
<td>CJNE A, # data, rel</td>
<td>Compare immediate to Acc and Jump if Not Equal</td>
<td>3</td>
<td>24</td>
</tr>
</tbody>
</table>

#### Instructions that Affect Flag Settings

<table>
<thead>
<tr>
<th>Instruction</th>
<th>C</th>
<th>OV</th>
<th>AC</th>
<th>Flag</th>
<th>Instruction</th>
<th>C</th>
<th>OV</th>
<th>AC</th>
<th>Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>CLR C</td>
<td>O</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>O</td>
</tr>
<tr>
<td>ADDC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>CPL C</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>C</td>
</tr>
<tr>
<td>SUBB</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>ANL C, bit</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>ANL C, bit</td>
</tr>
<tr>
<td>MUL</td>
<td>O</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>O</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DIV</td>
<td>O</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>O</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DA</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>RRC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>MOV C, bit</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>MOV C, bit</td>
</tr>
<tr>
<td>RLC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>CJNE</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>CJNE</td>
</tr>
<tr>
<td>SETB C</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

(1) Note that operations on SFR byte address 208 or bit addresses 209-215 (i.e., the PSW or bits in the PSW) will also affect flag settings.

Note on instruction set and addressing modes:

- **Rn** — Register R7–RO of the currently selected Register Bank.
- **direct** — 8-bit internal data location’s address. This could be an Internal Data RAM location (0–127) or a SFR [i.e., I/O port, control register, status register, etc. (128–255)].
- **@Ri** — 8-bit internal data RAM location (0–255) addressed indirectly through register R1 or R0.
- **#data** — 8-bit constant included in instruction.
- **#data 16** — 16-bit destination address. Used by LCAL and LJMP. A branch can be anywhere within the 64K-byte Program Memory address space.
- **addr 16** — 11-bit destination address. Used by LCAL and LJMP. The branch will be within the same 2K-byte page of program memory as the first byte of the following instruction.
- **rel** — Signed (two’s complement) 8-bit offset byte. Used by SJMP and all conditional jumps. Range is −128 to +127 bytes relative to first byte of the following instruction.
- **bit** — Direct Addressed bit in Internal Data RAM or Special Function Register.
Annexure -B

8051 Special Function Registers:

1. Timer Mode Control Register (TMOD):

TMOD can be considered to be two duplicate 4-bit registers, each of which controls the action of one of the timers. The “Timer” or “Counter” function is selected by control bits C/T, and in different operating modes, which are selected by bit-pairs (M1, M0) in TMOD.

<table>
<thead>
<tr>
<th>GATE</th>
<th>C / T</th>
<th>M1</th>
<th>M0</th>
<th>GATE</th>
<th>C / T</th>
<th>M1</th>
<th>M0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer 1</td>
<td></td>
<td></td>
<td></td>
<td>Timer 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GATE</td>
<td>C / T</td>
<td>M1</td>
<td>M0</td>
<td>GATE</td>
<td>C / T</td>
<td>M1</td>
<td>M0</td>
</tr>
<tr>
<td>Gating control when set. Counter “x” is enabled only while “INTx” pin is high and “TRx” control pin is set. When cleared, Timer “x” is enabled whenever “TRx” control bit is set.</td>
<td>Timer or Counter Selector cleared for Timer operation (input from internal system clock.) Set for Counter operation (input from “Tx” input pin).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C/T</td>
<td>OPERATION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0</td>
<td>13-bit Timer/Counter 5-bits of “TLx” and 8-bits of “THx” are used.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 1</td>
<td>16-bit Timer/Counter 8-bits of “TLx” and 8-bits of “THx” are cascaded.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 0</td>
<td>8-bit auto-reload Timer/Counter “THx” holds a value which is to be reloaded into “TLx” each time it overflows.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 0</td>
<td>(Timer 0) TL0 is an 8-bit Timer/Counter controlled by the standard Timer 0 control bits. TH0 is an 8-bit timer only controlled by Timer 1 control bits. Timer/Counter 1 stopped.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. Interrupt Enable (IE) Register:

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>EA</td>
<td>x</td>
<td>x</td>
<td>ES</td>
<td>ET1</td>
<td>EX1</td>
<td>ET0</td>
<td>EX0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>EA</td>
<td>Enable All. If 0, Disables all interrupts and no interrupt is acknowledged. If 1, each interrupt can be individually enabled or disabled by programming appropriate bit.</td>
</tr>
<tr>
<td>x</td>
<td>Reserved</td>
</tr>
<tr>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>ES</td>
<td>Enable Serial Interrupt. If 1, enables TI or RI to generate interrupt.</td>
</tr>
<tr>
<td>ET1</td>
<td>Enable Timer 1 interrupt. If 1, Enables the TF1 to generate the interrupt.</td>
</tr>
<tr>
<td>EX1</td>
<td>Enable External interrupt 1. If 1, Enables the INT1 to generate the interrupt.</td>
</tr>
<tr>
<td>ET0</td>
<td>Enable Timer 0 interrupt. If 1, Enables the TF0 to generate the interrupt.</td>
</tr>
<tr>
<td>EX0</td>
<td>Enable External interrupt 0. If 1, Enables the INT0 to generate the interrupt.</td>
</tr>
</tbody>
</table>
3. Timer Control Register (TCON):

TCON has control bits and flags for the timers in the upper nibble, and control bits and flags for the external interrupts in lower nibble.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Symbol</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCON.7</td>
<td>TF1</td>
<td>Timer 1 overflow flag. Set by hardware on Timer/Counter overflow. Cleared by hardware when processor vectors to interrupt routine, or clearing the bit in software.</td>
</tr>
<tr>
<td>TCON.6</td>
<td>TR1</td>
<td>Timer 1 Run control bit. Set/cleared by software to turn Timer/Counter on/off.</td>
</tr>
<tr>
<td>TCON.5</td>
<td>TF0</td>
<td>Timer 0 overflow flag. Set by hardware on Timer/Counter overflow. Cleared by hardware when processor vectors to interrupt routine, or by clearing the bit in software.</td>
</tr>
<tr>
<td>TCON.4</td>
<td>TR0</td>
<td>Timer 0 Run control bit. Set/cleared by software to turn Timer/Counter on/off.</td>
</tr>
<tr>
<td>TCON.3</td>
<td>IE1</td>
<td>Interrupt 1 Edge flag. Set by hardware when external interrupts edge detected. Cleared when interrupt processed.</td>
</tr>
<tr>
<td>TCON.2</td>
<td>IT1</td>
<td>Interrupt 1 type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.</td>
</tr>
<tr>
<td>TCON.1</td>
<td>IE0</td>
<td>Interrupt 0 Edge flag. Set by hardware when external interrupts edge detected. Cleared when interrupt processed.</td>
</tr>
<tr>
<td>TCON.0</td>
<td>IT0</td>
<td>Interrupt 0 Type control bit. Set/cleared by software to specify falling edge/low Level triggered external interrupts.</td>
</tr>
</tbody>
</table>

4. Interrupt Priority (IP) Register:

Each source of the interrupt can be individually programmed to be in either of the two priority levels. The priorities can be assigned to each interrupt by programming appropriate bits in the SFR Interrupt Priority Register.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>Reserved</td>
</tr>
<tr>
<td>PS</td>
<td>Priority of Serial Interrupt. If 1, Priority of Serial Interrupt is higher</td>
</tr>
<tr>
<td>PT1</td>
<td>Priority of Timer 1 interrupt. If 1, Priority of Timer 1 interrupt is higher</td>
</tr>
<tr>
<td>PX1</td>
<td>Priority of External interrupt 1. If 1, Priority of the INT1 is higher</td>
</tr>
<tr>
<td>PT0</td>
<td>Priority of Timer 0 interrupt. If 1, Priority of Timer 0 Interrupt is higher</td>
</tr>
<tr>
<td>PX0</td>
<td>Priority of External interrupt 0. If 1, Priority of the INT0 is higher</td>
</tr>
</tbody>
</table>
5. Serial Port Control Register (SCON):

The serial port control and status register is the Special Function Register SCON. This register contains not only the mode selection bits, but also the 9th data bit for transmit and receive (TB8 and RB8) and the serial port interrupt bits (TI and RI).

<table>
<thead>
<tr>
<th>SM0</th>
<th>SM1</th>
<th>Mode</th>
<th>Description</th>
<th>Baud Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Shift register</td>
<td>fosc / 12</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>8-bit UART</td>
<td>Variable</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>9-bit UART</td>
<td>fosc / 64 or fosc /32</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>9-bit UART</td>
<td>variable</td>
</tr>
</tbody>
</table>

Where SM0, SM1 specify the serial port mode, as follows:

- **SM2**: Enables the multiprocessor communication feature in Modes 2 and 3. In Mode 2 or 3, if SM2 is set to 1, then RI will not be activated if the received 9th data bit (RB8) is 0. In Mode 1, if SM2=1 then RI will not be activated if a valid stop bit was not received. In Mode 0, SM2 should be 0.

- **REN**: Enables serial reception. Set by software to enable reception. Clear by software to disable reception.

- **TB8**: The 9th data bit that will be transmitted in Modes 2 and 3. Set or clear by software as desired.

- **RB8**: In Modes 2 and 3, is the 9th data bit that was received. In Mode 1, it SM2=0, RB8 is the stop bit that was received. In Mode 0, RB8 is not used.

- **TI**: Transmit interrupt flag. Set by hardware at the end of the 8th bit time in Mode 0, or at the beginning of the stop bit in the other modes, in any serial transmission. Must be cleared by software only.

- **RI**: Receive interrupt flag. Set by hardware at the end of the 8th bit time in Mode 0, or halfway through the stop bit time in the other modes, in any serial reception (except see SM2). Must be cleared by software only.

---

**Practice does not make perfect. Only perfect practice makes perfect.**
Annexure C

Architecture of MSP430 microcontroller

Figure: Internal Architecture of MSP 430

Addressing mode

All the addressing modes for the source operand and all four addressing modes for the destination operand can address the complete address space. The bit number show the content of the As resp. Ad mode bits.

<table>
<thead>
<tr>
<th>As</th>
<th>Ad</th>
<th>Addressing Mode</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>Register Mode</td>
<td>Rn</td>
<td>Register contents are operand</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>Indexed Mode</td>
<td>X(Rn)</td>
<td>(Rn + X) points to the operand. X is stored in the next word</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>Symbolic Mode</td>
<td>ADDR</td>
<td>(PC + X) points to the operand. X is stored in the next word. Indexed Mode X(PC) is used</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>Absolute Mode</td>
<td>&amp;ADDR</td>
<td>The word following the instruction contains the absolute address.</td>
</tr>
<tr>
<td>10</td>
<td>-</td>
<td>Indirect Register Mode</td>
<td>@Rn</td>
<td>Rn is used as a pointer to the operand</td>
</tr>
<tr>
<td>11</td>
<td>-</td>
<td>Indirect Autoincrement</td>
<td>@Rn+</td>
<td>Rn is used as a pointer to the operand. Rn is incremented afterwards</td>
</tr>
<tr>
<td>11</td>
<td>-</td>
<td>Immediate Mode</td>
<td>#N</td>
<td>The word following the instruction contains the immediate constant N. Indirect Autoincrement Mode @PC+ is used</td>
</tr>
</tbody>
</table>

Table 1: Addressing modes in MSP 430
Instruction Set of MSP430

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC(B) dst</td>
<td>dst + C → dst</td>
</tr>
<tr>
<td>ADD(B) src, dst</td>
<td>src + dst → dst</td>
</tr>
<tr>
<td>ADDC(B) src, dst</td>
<td>src + dst + C → dst</td>
</tr>
<tr>
<td>AND(B) src, dst</td>
<td>src .and. dst → dst</td>
</tr>
<tr>
<td>BIC(B) src, dst</td>
<td>.not.src .and. dst → dst</td>
</tr>
<tr>
<td>BIS(B) src, dst</td>
<td>src .or. dst → dst</td>
</tr>
<tr>
<td>BIT(B) src, dst</td>
<td>src .and. dst</td>
</tr>
<tr>
<td>BR dst</td>
<td>Branch to …</td>
</tr>
<tr>
<td>CALL dst</td>
<td>PC+2 → stack, dst → PC</td>
</tr>
<tr>
<td>CLR(B) dst</td>
<td>Clear destination</td>
</tr>
<tr>
<td>CLRC</td>
<td>Clear carry bit</td>
</tr>
<tr>
<td>CLRN</td>
<td>Clear negative bit</td>
</tr>
<tr>
<td>CLRZ</td>
<td>Clear zero bit</td>
</tr>
<tr>
<td>CMP(B) src, dst</td>
<td>dst - src</td>
</tr>
<tr>
<td>DADD(C) dst</td>
<td>dst + C → dst (decimal)</td>
</tr>
<tr>
<td>DADD(B) src, dst</td>
<td>src + dst + C → dst (decimal)</td>
</tr>
<tr>
<td>DEC(B) dst</td>
<td>dst - 1 → dst</td>
</tr>
<tr>
<td>DEC(D) dst</td>
<td>dst - 2 → dst</td>
</tr>
<tr>
<td>DINT</td>
<td>Disable interrupt</td>
</tr>
<tr>
<td>EINT</td>
<td>Enable interrupt</td>
</tr>
<tr>
<td>INC(B) dst</td>
<td>Increment destination, dst+1 → dst</td>
</tr>
<tr>
<td>INCD(B) dst</td>
<td>Double-Increment destination, dst+2 → dst</td>
</tr>
<tr>
<td>INV(B) dst</td>
<td>Invert destination</td>
</tr>
<tr>
<td>JCEJHS Label</td>
<td>Jump to Label if Carry-bit is set</td>
</tr>
<tr>
<td>JEQZ Label</td>
<td>Jump to Label if Zero-bit is set</td>
</tr>
<tr>
<td>JEW Label</td>
<td>Jump to Label if (N XOR V) = 0</td>
</tr>
<tr>
<td>JL Label</td>
<td>Jump to Label if (N XOR V) = 1</td>
</tr>
<tr>
<td>JMP Label</td>
<td>Jump to Label unconditionally</td>
</tr>
<tr>
<td>JN Label</td>
<td>Jump to Label if Negative-bit is set</td>
</tr>
</tbody>
</table>

Legend:
0 Status bit always cleared
x Status bit cleared or set on results
* Status bit not affected
* Emulated Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JNC/UZ Label</td>
<td>Jump to Label if Carry-bit is reset</td>
</tr>
<tr>
<td>JNE/UZ Label</td>
<td>Jump to Label if Zero-bit is reset</td>
</tr>
<tr>
<td>MOV(B) src, dst</td>
<td>src → dst</td>
</tr>
<tr>
<td>NOP</td>
<td>No operation</td>
</tr>
<tr>
<td>POP(B) dst</td>
<td>Item from stack, SP+2 → SP</td>
</tr>
<tr>
<td>PUSH(B) src</td>
<td>SP - 2 → SP, src ← @SP</td>
</tr>
<tr>
<td>RET</td>
<td>Return from interrupt</td>
</tr>
<tr>
<td>RET</td>
<td>Return from subroutine</td>
</tr>
<tr>
<td>RLA(B) dst</td>
<td>Rotate left arithmetically</td>
</tr>
<tr>
<td>RLC(B) dst</td>
<td>Rotate left through carry</td>
</tr>
<tr>
<td>RRA(B) dst</td>
<td>MSB ← MSB … LSB → C</td>
</tr>
<tr>
<td>RRC(B) dst</td>
<td>C ← MSB … LSB → C</td>
</tr>
<tr>
<td>SBC(B) dst</td>
<td>Subtract carry from destination</td>
</tr>
<tr>
<td>SETC</td>
<td>Set carry bit</td>
</tr>
<tr>
<td>SETN</td>
<td>Set negative bit</td>
</tr>
<tr>
<td>SETZ</td>
<td>Set zero bit</td>
</tr>
<tr>
<td>SUB(B) src, dst</td>
<td>dst - src + 1 → dst</td>
</tr>
<tr>
<td>SWAPB dst</td>
<td>swap bytes</td>
</tr>
<tr>
<td>SXT dst</td>
<td>Bit7 → Bit8 … Bit15</td>
</tr>
<tr>
<td>TST(B) dst</td>
<td>Test destination</td>
</tr>
<tr>
<td>XOR(B) src, dst</td>
<td>src .xor. dst → dst</td>
</tr>
</tbody>
</table>

Legend:
0 The Status Bit is cleared
x The Status Bit is affected
* The Status Bit is not affected
* Emulated Instructions
APPENDIX D

MCU 8051 IDE

MCU 8051 IDE is a new modern graphical integrated development environment for microcontrollers based on 8051. For those who believe 8051 is a great piece of technology this IDE is a new way how to see and feel these still famous microcontrollers. **MCU 8051 IDE is noncommercial open-source** software primarily for Microsoft Windows and GNU/Linux. Supported programming languages are C language and assembly. It has its own assembler and support for 2 external assemblers. For C language it uses SDCC compiler. There are packages for various Linux distributions (.rpm and .deb), and installer for Microsoft Windows. This IDE contains simulator, source code editor, assembler, HW programmer and much other tools. Simulator supports over 79 MCU primarily from Atmel. There is also support for simple hardware simulation (like LEDs, keys, etc.).
List of major Features

- Advanced simulator with support for more than 79 MCUs.
- Built-in optimizing macro assembler
- Support for C language
- Simple external hardware simulation
- Interactive help for assembly language
- Interrupt monitor & editor
- Advanced source code editor
  - Syntax highlight
  - Syntax validation
  - Pop-up based auto-completion
  - Editor command line Interactive map of SFR
- Tools for converting between various data files related to MCUs
- Editor of bit addressable area in the simulated MCU
- Stopwatch timer
- Special calculator optimized for use with 8051
- Assembler symbol viewer
- Base converter
- 8 segment display editor
- Hexadecimal editor
- Interactive ASCII chart
- Simple graphical notepad
- Various information windows, tool tips, legends
- IO Ports monitor

Virtual Hardware Simulator

MCU 8051 IDE simulator is also equipped with a few simulated simple hardware devices, which can be connected to the simulated MCU. These virtual hardware components are intended primarily to offer a better insight into programs interacting with things like LEDs or keys.
Fig: Virtual Hardware Examples