||Jai Sri Gurudev||

ADICHUNCHANAGIRI UNIVERSITY

BGS INSTITUTE OF TECHNOLOGY

BG Nagara – 571448 (Bellur Cross)
Nagamangala Taluk, Mandya District.

MICROCONTROLLER LABORATORY MANUAL
18ECL47

For
IV Semester B.E.
2019-2020

DEPARTMENT OF
ELECTRONICS AND COMMUNICATION ENGINEERING

Prepared by:

1. Dr. Naveen B, Asso. Prof
2. Mrs. RAMYA K, Asst. Prof
3. Mr. GOUTHAM V, Asst. Prof

Approved by:

Head of Department

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENFINEERING

VISION:

To develop high quality engineers with technical knowledge, skills and ethics in the area of Electronics and Communication Engineering to meet industrial and societal needs.

MISSION:

1. To provide high quality technical education with up-to-date infrastructure and trained human resources to deliver the curriculum effectively in order to impart technical knowledge and skills.
2. To train the students with entrepreneurship qualities, multidisciplinary knowledge and latest skill sets as required for industry, competitive examinations, higher studies and research activities.
3. To mould the students into professionally-ethical and socially-responsible engineers of high character, team spirit and leadership qualities.

PROGRAM EDUCATIONAL OBJECTIVES (PEO's):

After 3 to 5 years of graduation, the graduates of Electronics and Communication Engineering will;

1. Engage in industrial, teaching or any technical profession and pursue higher studies and research.
2. Apply the knowledge of Mathematics, Science as well as Electronics and Communication Engineering to solve social engineering problems.
3. Understand, Analyze, Design and Create novel products and solutions.
4. Display professional and leadership qualities, communication skills, team spirit, multidisciplinary traits and lifelong learning aptitude.

MICROCONTROLLER LAB SYLLABUS

Course Learning Objectives:

This laboratory course enables students to

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

Laboratory Experiments

I. AL PROGRAMMING

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

II. INTERFACING

1. Write a C program to rotate Stepper motor control interface to 8051.
2. Write a C program to rotate DC motor control interface to 8051.
3. Write a C program for Elevator interface to 8051.
4. Write a C program for SEVEN SEGMENT DISPLAY.
5. Generate different waveforms Square, Triangular, using DAC interface to 8051; change the frequency and amplitude.

Beyond Syllabus:

1. Write an ALP to generate the Delay.
2. Generate Sawtooth waveforms using DAC interface to 8051; change the frequency and amplitude.

Course Outcomes:

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

1. Analyze 8051 assembly level programs to perform data transfer, arithmetic, Boolean and logical operations.
2. Analyze 8051 assembly level programs to perform counter operation along with conditional call and return operation.
3. Analyze 8051 assembly level programs to perform code conversion operation like BCD, ASCII, decimal and Hex operation.
4. Demonstrate the interfacing of 8051 C Programs with Stepper Motor, DC Motor, Elevator Interface, and 7 segment displays.
5. Demonstrate the interfacing of 8051 C Programs to generate different square, Triangular waveform using DAC.

ALGORITHM

1. Initialize registers to hold count data & also the source & destination addresses
2. Get data from source location into accumulator and transfer to the destination Location word by word.
3. Decrement the count register and repeat step 2 till count is zero.

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

RESULT :

Content of source array before execution

20h	01
21h	02
22h	03
23h	04
24h	05
25h	06
26h	07
27h	08
28h	09
29h	0A

Content of source array before execution

30h	00
31h	00
32h	00
33h	00
34h	00
35h	00
36h	00
37h	00
38h	00
39h	00

Content source array after execution

20h	01
21h	02
22h	03
23h	04
24h	05
25h	06
26h	07
27h	08
28h	09
29h	0A

Content destination array after execution

30h	01
31h	02
32h	03
33h	04
34h	05
35h	06
36h	07
37h	08
38h	09
39h	0A

1. **Write an ALP to transfer of block of data from one location to another location.**

```
        Org 000h
        mov r2,#0ah        //count
        mov r0,#20h        //source address
        mov r1,#30h        //destination address
loop1:  mov a,@r0
        mov @r1,a
        inc r0
        inc r1
        djnz r2, loop1
here:   sjmp here
        end
```

ALGORITHM

1. Initialize registers to hold count data & also the source & destination of last addresses.
2. Get data from source location into accumulator and transfer to the destination location.
3. Decrement the count register and repeat step 2 till count is zero.

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

RESULT :

Content of source array before execution

20h	01
21h	02
22h	03
23h	04
24h	05
25h	06
26h	07
27h	08
28h	09
29h	0A

Content of source array before execution

30h	00
2Fh	00
2Eh	00
2Dh	00
2Ch	00
2Bh	00
2Ah	00
29h	00
28h	00
27h	00

Content source array after execution

20h	01
21h	02
22h	03
23h	04
24h	05
25h	06
26h	07
27h	08
28h	09
29h	0A

Content destination array after execution

30h	0A
2Fh	09
2Eh	08
2Dh	07
2Ch	06
2Bh	05
2Ah	04
29h	03
28h	02
27h	01

2. Write an ALP to transfer of block of data from one location to other location with overlap.

```
        org 00h
        mov r2,#0ah          //count
        mov r0,#29h          //source address
        mov r1,#30h          //destination address
loop1:  mov a,@r0
        mov @r1,a
        dec r0
        dec r1
        djnz  r2,loop1
here:   sjmp here
        end
```

ALGORITHM

1. Initialize registers to hold count data (array size) & also the source & destination addresses.
2. Get data from source location into accumulator and save in a register.
3. Get data from the destination location into accumulator.
4. Exchange the data at the two memory locations.
5. Decrement the count register and repeat from step 2 to 4 till count is zero.

RESULT :

Content of source array before execution

50h	01
51h	02
52h	03
53h	04
54h	05
55h	06
56h	07
57h	08
58h	09
59h	0A

Content of source array before execution

70h	11
71h	22
72h	33
73h	44
74h	55
75h	66
76h	77
77h	88
78h	99
79h	AA

Content source array after execution

50h	11
51h	22
52h	33
53h	44
54h	55
55h	66
56h	77
57h	88
58h	99
59h	AA

Content destination array after execution

70h	01
71h	02
72h	03
73h	04
74h	05
75h	06
76h	07
77h	08
78h	09
79h	0A

3. Write an ALP to perform Exchange of block of data between two memory location.

```
        org 00h
        mov r2,#0ah
        mov r0,#50h
        mov r1,#70h
loop1:  mov a,@r0
        xch a,@r1
        mov @r0,a
        inc r0
        inc r1
        djnz  r2,loop1
here:   sjmp here
        end
```

ALOGORITHM:

1. Take lower byte of 16 bit number on register A add with lower byte of second number.
2. Store the result in some register
3. Take the higher byte of 16 bit number on register A add with carry with higher byte of second number.
4. Store the result in some register

RESULT :

Content of reference registers - before execution	**Content of reference registers -after execution**
[A] = 3Ch	[A] =BAh
[A] +[DF] = 1Bh	[A] + [04h] = BFh
[A] = 1Bh	[A] = [BFh]
[R0] = 1Bh	[A] = BFh

4.Write an ALP to perform 16 bit addition.

```
        org 00h
        clr c
        mov a,#03Ch
        add a,#0DFh
        mov r0,a
        mov a,#0BAh
        addc a,#04h
        mov r1,a
here: sjmp here
        end
```

ALOGORITHM:

1. Take lower byte of 16 bit number on register A add with lower byte of second number
2. Store the result in some register
3. Take the higher byte of 16 bit number on register A add with carry with higher byte of second number.
4. Store the result in some register.

RESULT :

Content of reference registers - before execution	**Content of reference registers - after execution**
[A] = 00h	[A] =75h [A] - [48h] = 2dh [A] = 2dh [A] = 2dh

5.Write an ALP to perform 16 bit Subtraction.

```
        org 00h
        clr c
        mov a,#53h
        subb a,#58h
        mov r0,a
        mov a,#48h
        subb a,#22h
        mov r1,a
here:   sjmp here
        end
```

ALGORITHM

1. Store the elements of the array from the address.
2. Store the length of the array in and set it as counter.
3. Register is loaded with starting address of the array.
4. Store the first number of the array in b (b is assigned to hold the largest number).
5. Increment Register.
6. Subtract the number pointed by Register from the contents of b (to compare whether the next array element is larger than the one in b).
7. If the element pointed by Register is larger then load the larger number into b.
8. Decrement the counter and repeat steps through 5 until the counter becomes 0.

RESULT :

Content of source array before execution

20h	01
21h	02
22h	03
23h	04
24h	05
25h	06
26h	07
27h	08
28h	09
29h	0A

Content of reference registers -after Execution

R4 =0Ah

6. Write an ALP to find largest number in an array

```
        org 00h
        mov r5,#0ah
        mov r0,#20h
        mov b,#00h
repeat:mov a,@r0
       cjne a,b,ne1
  ne1:jc loop1
        mov b,a
        inc r0
        djnz r5,repeat
        sjmp exit
loop1:inc r0
        djnz r5,repeat
  exit:mov a,b
        mov r4,a
here:  sjmp here
        end
```

ALGORITHM

1. Store the condition x in r1.
2. Load the first and second numbers to A and B registers respectively
3. Compare the contents of r1 and perform the operations add, sub, etc accordingly. Store the result present in A and B registers to the appropriate memory locations.

RESULT:

a) AND OPERATION

Content of reference registers after execution

[R0] = 34h
[A] = 0Fh
[P0] = 04h

b) OR OPERATION

Content of reference registers after execution

[R0] = 34h
[A] = 0F0h
[P1] = F4h

c) XOR OPERATION

Content of reference registers after execution

[R0] = 34h
[A] = 0Fh
[P2] = 3bh

d) 1'S COMPLEMENTS

Content of reference registers after execution

[A] + [R0] = 34h
[P0] = cbh

e) 2'S COMPLEMENTS

After Execution

[A] + [R0] = 34h
[P0] = cch

7. Write an ALP to perform the following logical operation.
a) AND b) OR c) XOR d) Complements

```
        org 00h
        mov r0,#34h
        call and1
        call or1
        call xor1
        call comp
here: sjmp here
 and:  mov a,#0fh
        anl a,r0
        mov p0,a
        ret
  or1: mov a,#0fh
        orl a,r0
        mov p1,a
        ret
 xor1: mov a,#0fh
        xrl a,r0
        mov p2,a
        ret
comp: mov a,r0
        cpl a
        add a,#01h
        mov p3,a
        ret
        end
```

ALOGORITHM:

1. Take the given number in some register.
2. Initiate two register to count zeros and ones.
3 .Take the given number in register A.
4. Rotate the content of A either towards left or right through carry flag.
5. If the carry flag is one increment the content of one's register by one, else zero register
6. Repeat the above steps for 8 times.

RESULT:

Content of reference registers after execution

1. [A] = 31h
   [R0] = 05h
   [P0] = 03h

2. [A] = 99h
   [R0] = 04h
   [P0] = 04h

3. [A] = 03h
   [R0] = 02h
   [P0] = 06h

8. Write an ALP to find the number of 1's and 0's in a byte.

```
        org 00h
        mov r0,#00h
        mov r1,#00h
        mov r2,#08h
        mov a,#31h
repeat: rrc a
        jnc loop1
        inc r1
        sjmp exit
loop1: inc r0
  exit: djnz r2,repeat
here: sjmp here
        end
```

ALGORITHM:

1. Move 00 to r2 register
2. Increment the content of register by one
3. After each count call delay subroutine
4. Compare the content of r2 with 0FH
5. Reload r2 with 00 repeat step 2 as specified above.

RESULT:

	Content of reference registers before execution	**Content of registers after execution**
R2 =	00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h	00h 01h 02h 03h 04h 05h 06h 07h 08h 09h 0Ah 0Bh 0Ch 0Dh 0Eh 0Fh

9. Write an ALP to perform the 4 bit Up Counter operation.

```
        org 00h
        mov r2,#00h
   up: inc r2
       cjne r2,#0fh,loop1
        Acall delay
       mov r2,#00h
loop1: Acall delay
        sjmp up
delay : mov r0,#0ffh
l3:     mov r1,#0ffh
l2:     mov r3,#0ffh
l1:     djnz r3,l1
        djnz r1,l2
        djnz r0,l3
        ret
 here: sjmp here
        end
```

ALGORITHM:

1. Load initially r2 register with 0fh.
2. Decrement the content of register by one
3. After each count call delay subroutine
4. Campare the content of r2 with 00h
5. Reload r2 with 0fh repeat step 2 as specified above.

RESULT:

Content of reference registers before execution		Content of registers after execution	
R2 =	00h		0Fh
	00h		0Eh
	00h		0Dh
	00h		0Ch
	00h		0Bh
	00h		0Ah
	00h		09h
	00h		08h
	00h		07h
	00h		06h
	00h		05h
	00h		04h
	00h		03h
	00h		02h
	00h		01h
	00h		00h

10. Write an ALP to perform the 4 bit Down Counter operation.

```
        org 00h
        mov r2,#ofh
down:dec r2
        cjne r2,#00h,loop1
        Acall delay
        mov r2,#0fh
loop1: Acall delay
         sjmp down
delay : mov r0,#0ffh
l3:      mov r1,#0ffh
l2:      mov r3,#0ffh
l1:     djnz r3,l1
        djnz r1,l2
        djnz r0,l3
        ret
here: sjmp here
        end
```

ALGORITHM:

1. Move 00 to r2 register
2. Increment the content of register by one
3. After each count call delay subroutine
4. Compare the content of r2 with 0fh
5. If it is 0Fh decrement by 1.
6. When r2 register becomes 00h repeat step 2 as specified above.

RESULT:

Content of registers after execution

Up counter sequence	Down Count Sequence
00h	0Fh
01h	0Eh
02h	0Dh
03h	0Ch
04h	0Bh
05h	0Ah
06h	09h
07h	08h
08h	07h
09h	06h
0Ah	05h
0Bh	04h
0Ch	03h
0Dh	02h
0Eh	01h
0Fh	00h

11. Write an ALP to perform the 4 bit Up-Down Counter operation.

```
        org 00h
        mov r2,#00h
   up:inc r2
        cjne r2,#0fh,loop1
        sjmp down
loop1:Acall delay
        sjmp up
loop2:Acall delay
         sjmp down
down:dec r2
        cjne r2,#00h,loop2
        Acall delay
        sjmp up
delay : mov r0,#0ffh
l3:      mov r1,#0ffh
l2:      mov r3,#0ffh
l1:     djnz r3,l1
        djnz r1,l2
        djnz r0,l3
        ret
here: sjmp here
        end
```

Algorithm:

1. Move 00 to r2 register
2. Increment the content of register by one
3. After each count call delay subroutine
4. Compare the content of r2 with 0ffh
5. Repeat step 1

RESULT:

	Content of reference registers before execution	**Content of registers after execution**
R2 =	00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h	00h 01h 02h 03h 04h 05h 06h 07h 08h 09h 0Ah 0Bh 0Ch 0Dh 0Eh 0Fh 10h 11h 12h ---- ---- 0FFh

12. Write an ALP to perform the 8 bit Up Counter operation.

```
        org 00h
        mov r2,#00h
   up:inc r2
        Acall delay
       cjne r2,#0ffh,loop1
        Acall delay
       mov r2,#00h
loop1:sjmp up
delay : mov r0,#0ffh
l3:     mov r1,#0ffh
l2:     mov r3,#0ffh
l1:     djnz r3,l1
        djnz r1,l2
        djnz r0,l3
        ret
here: sjmp here
        end
```

ALGORITHM:

1. Move 0ff to r2 register
2. Increment the content of register by one
3. After each count call delay subroutine
4. Compare the content of r2 with 00h
5. Repeat step 1

RESULT:

Content of reference registers before execution		**Content of registers after execution**
R2 =	00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h 00h	0FFh 0FEh 0FDh 0FCh 0FBh 0FAh 0F9h 0F8h 0F7h 0F6h 0F5h 0F4h 0F3h 0F2h 0F1h ---- 00h

13. Write an ALP to perform the 8 bit Down Counter operation.

```
        org 00h
        mov r2,#offh
down:dec r2
        cjne r2,#00h,loop1
        mov r2,#0ffh
loop1:Acall delay
        sjmp down
delay : mov r0,#0ffh
l3:      mov r1,#0ffh
l2:      mov r3,#0ffh
l1:     djnz r3,l1
        djnz r1,l2
        djnz r0,l3
        ret
here: sjmp here
        end
```

ALGORITHM:

1. Move 00 to r2 register
2. Increment the content of register by one
3. After each count call delay subroutine
4. Compare the content of r2 with 0fh
5. If it is 0ffh decrement count value by 1.

RESULT:

Content of reference registers before execution	**Content of registers after execution**
00h	0FFh
01h	0FEh
02h	0FDh
03h	0FCh
04h	0FBh
05h	0FAh
06h	0F9h
07h	0F8h
08h	0F7h
09h	0F6h
0Ah	0F5h
0Bh	0F4h
0Ch	0F3h
0Dh	0F2h
0Eh	0F1h
0Fh	----

0FFh	00h

14. Write an ALP to perform the 8 bit Up-Down Counter operation.

```
        org 00h
        mov r2,#00h
   up: inc r2
        Acall delay
        cjne r2,#0ffh,loop1
        Acall delay
        sjmp down
loop1:sjmp up
loop2:sjmp down
down:dec r2
        Acall delay
        cjne r2,#00h,loop2
        Acall delay
        sjmp up
delay : mov r0,#0ffh
l3:     mov r1,#0ffh
l2:     mov r3,#0ffh
l1:     djnz r3,l1
        djnz r1,l2
        djnz r0,l3
        ret
here: sjmp here
        end
```

ALGORITHM:

1. load accumulator with the code to be test present in memory 50h.
2. perform logical AND between A and 0e0h and jump if no zero to check remaining lower 5 bits otherwise move A=00h and halt.
3. Rotate left along with carry 5 times and check each time carry is generated and if increment R1 and last check with 2 if equal move A=FFh otherwise A=00h

RESULT:

Content of reference registers before execution	**Content of registers after execution**
1. [50h] = 18	[A] = FFh
2. [50h] = 02	[A] = 00h

15. Write an ALP to find whether the given code is 2 out of 5 or not. If code is valid display FF or else display 00

```
        org 00h
         mov r0,#05h
        mov r2,#50h
         mov r1,#00h
         mov a,@r2
        anl a,#0e0h
        jnz loop2
        mov a,@r2
repeat:rlc a
        jnc loop1
        inc r1
loop1:djnz r0,repeat
        cjne r1,#02h,loop2
        mov a,#0ffh
        sjmp here
loop2:mov a,#00h
 here:  sjmp here
        end
```

ALGORITHM:

1. Initialize counter value as per the length of series to be generated and specify the memory location to store the generated series via a register.
2. Fill the first two memory location with 00h and 01h and decrement counter value by 2
3. Decrement memory location by 2 and load the value of memory in register1 and increment memory location by one and load the content of this location in register2 and perform addition operation between the two registers.
4. Increment memory location and put the result in this location.
5. After each addition and storage decrement counter and STEP 3 until counter becomes zero.
6. Halt the the program.

RESULT:

Content of reference memory array before execution

20h	00h
21h	01h
22h	01h
23h	02h
24h	03h
25h	05h
26h	08h
27h	13h
28h	21h
29h	34h
2Ah	55h
2Bh	89h

Content of array after execution

20h	00h
21h	01h
22h	01h
23h	02h
24h	03h
25h	05h
26h	08h
27h	0Dh
28h	15h
29h	22h
2Ah	37h
2Bh	59h

16. Write an ALP to generate Fibonacci Series with certain range.

```
        org 00h
        mov r2,#0ah
        mov r0,#20h
        mov @r0,#00h
        inc r0
        mov @r0,#01h
repeat: mov a,@r0
        dec r0
        add a,@r0
         inc r0
        inc r0
        mov @r0,a
        djnz r2,repeat
here: sjmp here
        end
```

ALGORITHM:

1. Load the memory content into Accumulator By using indirect addressing mode
2. Perform AND operation between Acc and immediate number 0Fh and add 30h to the accumulator and save the result in R3 register.
3. Repeate step 1
4. Perform AND operation between Accumulator and immediate number 0F0h and rotate left or right 4 times and ADD 30h to Acc and save the result in R4 register..

RESULT:

Content of reference registers before execution	**Content of registers after execution**
[A] = 35h	[A] = 35h [R3] = 35h [R4] = 33h

17. Write an ALP to convert packed BCD number to ASCII.

```
        org 00h
        mov r0,#40h
        mov  a,@r0
        mov r2,a
        anl a,#0fh
        orl a,#30h
        mov r3,a
        mov a,r2
        anl a,#0f0h
        swap a
        orl a,#30h
        mov r4,a
here: sjmp here
        end
```

ALGORITHM:

1. Initialize registers to hold count data (array size) & also the source & destination addresses.
2. Get data from source location into accumulator and ADD 30h to it and put in destination memory location.
3. Decrement counter and Increment source and destination memory location and repeat STEP 2 until counter becomes zero.
4. Halt the program.

RESULT:

Content of reference registers before execution

20h	00h	30h	00h
21h	01h	31h	00h
22h	05h	32h	00h
23h	09h	33h	00h
24h	08h	34h	00h

Content of registers after execution

20h	00h	30h	30h
21h	01h	31h	31h
22h	05h	32h	35h
23h	09h	33h	39h
24h	08h	34h	38h

18. Write an ALP to convert Decimal number to ASCII.

```
        org 00h
        mov r0,#20h
        mov r1,#30h
        mov r2,#05h
repeat:mov a,@r0
        orl a,#30h
        mov @r1,a
        inc r0
        inc r1
        djnz r2,repeat
here: sjmp here
        end
```

ALGORITHM:

1. Load one ASCII code into Acc and perform Logical XOR operation between Acc and immediate data 30h and store the result in register
2. Load other ASCII code into ACC and perform Logical XOR operation between Accumulator and immediate data 30h and rotate the result left or right 4 times and store the result in register
3. ADD the result obtain in STEP 1 and STEP 2
4. Halt the program.

RESULT:

Content of registers after execution

[A] = 32h
[R2] = 02h
[A] = 33h
[A] = 23h

19. Write an ALP to convert two ASCII digits into one equivalent packed BCD number.

```
        org 00h
        mov r0,#40h
        mov a,@r0
        xrl a,#30h
        mov r2,a
        mov a,#33h
        xrl a,#30h
        swap a
        add a,r2
here:   sjmp here
        end
```

ALGORITHM:

1. Initialize registers to hold count data (array size) & also the source & destination addresses.
2. Get data from source location into accumulator and perform logical XOR operation between ACC and immediate data 30h and store the result in destination address.
3. Decrement counter and increment source and destination addresses
4. Repeat STEP 2 and STEP 3 until counter becomes zero.
5. Halt the program.

RESULT:

Content of reference registers before execution

20h	30h	30h	00h
21h	32h	30h	00h
22h	33h	30h	00h
23h	36h	30h	00h
24h	38h	30h	00h

Content of registers after execution

20h	30h	30h	00d
21h	32h	31h	02d
22h	33h	32h	03d
23h	36h	33h	06d
24h	38h	34h	08d

20. Write an ALP to convert ASCII number to Decimal equivalent.

```
        org 00h
        mov r2,#05h
        mov r0,#20h
        mov r1,#30h
repeat: mov a,@r0
        xrl a,#30h
        mov @r1,a
        inc r0
        inc r1
        djnz r2,repeat
here: sjmp here
        end
```

ALGORITHM:

1. Load decimal data to be converted to hexa into Acc.
2. Perform division of Acc with 10H stored in B register.
3. perform addition of Acc and B register.
4.Store the result in Destination location.
5. Halt the the Program.

RESULT:

Content of reference registers before execution	**Content of registers after execution**
[A] = 25 in decimal [B] = 10h in hexadecimal	[A] = 19h

21. Write an ALP to convert a Decimal number to equivalent

Hexadecimal number.

```
        org 00h
        mov a,#25
        mov b,#10h
        div ab
        swap a
        add a,b
        mov r2,a
here: sjmp here
        end
```

ALGORITHM:

1. Load the data to be swapped into Acc and save this data in some other register for further manipulation.
2. Initialize two counter each one is loaded with 04h immediate value.
3. Perform AND operation between ACC and immediate value 0fh and perform rotate operation .
4. For every rotation decrement one counter and repeat rotation until counter becomes zero and save this result in Register(say R2).
5. Load Acc With the value Saved in STEP 1 and repeat Step 3
6. For every rotation decrement another counter and repeat rotation until other counter becomes zero and save this result in other Register(say R3)
7. ADD the register contents obtain in STEP 4 and STEP 6(i.e R2 and R3)
8. Halt the program.

RESULT:

Content of reference registers before execution	**Content of registers after execution**
[A] = 35h	[A] = 53h

22. Write an ALP to reverse a number without using SWAP instruction.

```
        org 00h
        mov r2,#04h
        mov r4,#04h
        mov r3,#35h
        mov a,r3
        anl a,#0fh
repeat1:rl a
        djnz r2,repeat1
        mov r1,a
        mov a,r3
        anl a,#0f0h
repeat2:rr a
        djnz r4,repeat2
        add a,r1
        mov r5,a
here: sjmp here
        end
```

ALGORITHM:

1. Load the value in Accumulator which should be performed cubic operation.
2. Load the same value in B register and perform multiplication of Accumulator and B register
3. Compare the content of r2 with 0fh
4. If it is 0Fh decrement by 1.

RESULT:

Content of reference registers before execution	Content of registers after execution
[A] = FFh	[R4] = FFh
[B] = FFh	[R6] = 02h
	[R7] = Fdh

23. Write an ALP to find cube of 8-bit number.

```
        org 00h
        mov a,#0ffh
        mov b,#0ffh
        mov r2,b
        mul ab
        mov r3,b
        mov b,r2
        mul ab
        mov r4,a
        mov r5,b
        mov b,r3
        mov a,r2
        mul ab
        add a,r5
        mov r6,a
        mov r7,b
here:   sjmp here
        end
```

ALGORITHM:

1. Load the Accumulator with the value to be converted into decimal.
2. Load B register with the immediate value 0Ah.
3. Perform division operation between Accumulator and B register.
4. Save the value of B register obtain after division in some register (say R2)
5. Load again B register with immediate value 0Ah.
6. Again perform division between Accumulator and B register.
7. Save the values of B register and Accumulator in some registers (say R3 and R4)
8. The values present in R2, R3 and R4 gives decimal numbers.

RESULT:

Content of reference registers before execution	**Content of registers after execution**
[A] = FFh [B] = 0Ah	[A] = FFh [R4] = 02 [R3] = 05 [R2] = 05

24. Write an ALP to convert Hexadecimal to decimal conversion.

```
        org 00h
        mov a,#0ffh
        mov b,#0ah
        div ab
        mov r2,b
        mov b,#0ah
        div ab
        mov r3,b
        mov r4,a
here: sjmp here
        end
```

INTERFACING

1. Write an 8051 C Program to interface Elevator.

```
#include<reg52.h>
#define DATA_BUS P2  //CONSIDER PORT2 AS DATA BUS
#define ADD_BUS  P0   //CONSIDER PORT0 AS ADDRESS BUS

        void main(void)
{
        unsigned char cur_flr, next_flr,temp;
        //VARIABLES DECLARATION
        void delay_msec(unsigned int count); //DELALY ROUTINE
        void lssd(unsigned char temp);
        //LSSD DISPLAY ROUTINE

        cur_flr                 = 0;
        //INITIALIZE CURRENT FLOOR WITH 00H
        next_flr        = cur_flr;              //AT THE BEGINING
      ASSUME CURRENT FLOOR IS EAUAL TO NEXT FLOOR
        DATA_BUS        = 0xC0;
        //PLACE 0C0H(SEVEN SEGMENT ECODE FALUE FOR
                        ZERO) ON DATA BUS TO DISPLAY 0
        ADD_BUS         = 0x01;
        //ENABLE THE FIRST SEVEN SEGMENT DISPLAY TO
                                DISPLAY 0 ON IT
        ADD_BUS         = 0xFF;
        //DISABLE ALL PERIPHERAL DEVICES.

                while(1)
        {
                DATA_BUS        = 0xFF;
        //CONFIGURE DATA BUS AS INPUT PORT TO READ
          FLOOR VALUE
                ADD_BUS  = 0x20;        //ENABLE THE ELEVATOR
                                TO READ FLOOR VALUE
                temp                    = DATA_BUS;     //READ THE
                                                FLOOR VALUE
                ADD_BUS  = 0XFF;        //DISABLE ALL
                                        PERIPHERAL DEVICE.
                        if(temp         != 0xff)        //IF FLOOR IS
                                        SELECTED
```

```
{
        switch(temp)
    {
case 0xFE:   //IS FLOOR0       CONNECTED TO P2.0
next_flr = 0x00;
break;
case 0xEF:   //IS FLOOR1 CONNECTED TO P2.4
next_flr = 0x01;
break;
case 0xFD:   //IS FLOOR2 CONNECTED TO P2.1
next_flr = 0x02;
break;
case 0xDF:   //IS FLOOR3 CONNECTED TO P2.5
next_flr = 0x03;
break;
case 0xBF:   //IS FLOOR4 CONNECTED TO P2.6
next_flr = 0x04;
break;
case 0xFB:   //IS FLOOR5 CONNECTED TO P2.2
next_flr = 0x05;
break;
case 0xF7:   //IS FLOOR6 CONNECTED TO P2.3
next_flr = 0x06;
break;
case 0x7F:   //IS FLOOR7 CONNECTED TO P2.7
next_flr = 0x07;
    }
}
while(cur_flr != next_flr) //CONTINURE UNTIL
CURRENT FLOOR IS NOT EQUAL TO NEXT FLOOR VALUE
{
        if(cur_flr < next_flr)       //IF CURRENT
FLOOR VALUE IS LESS THAN NEXT FLOOR VALUE
    {
        cur_flr = cur_flr + 1;  //THEN
INCREMENT THE CURRENT FLOOR VALUE
    }
        else
    //IF CURRENT FLOOR VALUE IS GREATER
THAN NEXT FLOOR VALUE
    {
```

```
                cur_flr = cur_flr - 1;        //THEN DECREMENT THE
CURRENT FLOOR VALUE
                }
                delay_msec(500);
        //DELAY ROUTINE WITH 500msec DEALY
                lssd(cur_flr);
        //DISPLAY THE CURRENT FLOOR VALUE.
                }
        }
}


/*-------------------------------------------------------------------------
-----
FUNCTION NAME           : SEVEN SEGMENT DISPLAY
ROUTINE

DESCRIBITION   :        IN THIS FUNCTION, BCD VALUE
FROM 0 TO 7 CAN BE DISPLAYED ON THE
                                                SELECTED
SEVEN SEGMENT DISPLAY
-------------------------------------------------------------------------
---*/

        void lssd(unsigned char cur_flr)
{
        unsigned char
bcd[8]={0xC0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8}; //ENCODED
SEVEN SEGMENT ARRAY FROM 0 TO 7
        unsigned char i = 0;
                while(1)
        {
                if(i == cur_flr)      //SELECT POSITION OF
SEVEN SEGMENT ENDCODED VALUE CORRESPONDING
CUR_FLR VALUE
                {
                        break;
                }
                i++;
        }
        DATA_BUS = bcd[i];     //PLACED SEVEN SEGMENT
ENCODED VALUE ON THE DATA BUS
```

```
        ADD_BUS  = 0x01;              //ENABLE FIRST SEVEN
SEGMENT DISPLAY TO DISPLAY CUR_FLR VALUE
        ADD_BUS  = 0xff;              //DISABLE ALL
PERIPHERAL DEVCIES
}

/*-------------------------------------------------------------------------
-----
FUNCTION NAME              : TIMER0 AS DELAY
GENERATOR

DESCRIBITION   :       IN THIS FUNCTION, TIMER/COUNTER0
IS CONFIGURED AS DELAY GENERATOR WITH

        RESOLUTION OF 1 msec.
                                                        NOTE:
CLOCK/CYCLE = 6
                                                        TH0TL0 =
65536 - ((11.0592 X 10^6) X DELAY
RESOLUTION)/(CLOCK/CYCLE)
                                                        IF DELAY
RESOLUTION = 1 msec. AND CLOCK/CYBLE=6
                                                        THEN
TH0TL0 = 65536 - (11.0592 X 10^6) X 1X10^3)/6       = F8CCH
-------------------------------------------------------------------------
---*/
        void delay_msec(unsigned int count)
{
        unsigned int i;
        TMOD = 0X01;
        //CONFIGURE TIMER/COUNTER0 AS TIMER FOR
MODE1(16-BIT COUNT)
        TR0  = 1;
        //START TIMER0
        for(i = 0; i < count; i++)  //LOOP AS LONG AS REQUIRED
DELAY IS ATTAINED
        {
                TH0 = 0XF8;
        //ASSIGN VALUE TO TIMER0 REGISTER TO GENERATE
1 mSEC DELAY
                TL0 = 0XCC;
```

```
                while(!TF0);
        //LOOP HERE UNTIL TIMER0 OVERFLOW FLAG GETS
SET
                TF0 = 0;
        //CLEAR TIMER0 OVERFLOW FLAG TO CHECK NEXT
OVERFLOW
        }
        TR0 = 0;
        //STOP TIMER0
}
```

2. Write an 8051 C Program to rotate the Stepper motor in clockwise & anticlockwise direction.

```
#include <reg52.h>

        void main(void)
{
        char i,full_step[4] = {0x0A,0x09,0x05,0x06};            //Declare
Full-Step Array
        void delay_msec(unsigned int count);
        //Delay Routine Declaration with 1msec resolution
                while(1)

        //Execute while body for infinite time
        {
                        for(i = 3;i >= 0; i--)
                                        // select pulse location of Full-Step
Array
                {
                        P2      = full_step[i];
                                                        //Place Full-Step Pulse
on Data Bus
                        P0  = 0x07;
                                                                        //Place 07H
SM address on Address Bus to latch Pulse available on Data Bus
                        P0  = 0XFF;
                                                                        //Disable
SM latch
                        delay_msec(3);
                                                                //Call Delay
Routine to generate delay.
                }
        }
}

/*-------------------------------------------------------------------------
-----
FUNCTION NAME        : TIMER0 AS DELAY GENERATOR

DESCRIBITION  :         IN THIS FUNCTION, TIMER/COUNTER0
IS CONFIGURED AS DELAY GENERATOR WITH
```

```
                                                        RESOLUTION
OF 1 msec.
                                                        NOTE:
CLOCK/CYCLE = 6
                                                        TH0TL0 = 65536 -
((11.0592 X 10^6) X DELAY RESOLUTION)/(CLOCK/CYCLE)
                                                        IF DELAY
RESOLUTION = 1 msec. AND CLOCK/CYCLE=6
                                                        THEN TH0TL0 =
65536 - (11.0592 X 10^6) X 1X10^3)/6        = F8CCH
-------------------------------------------------------------------------------
---*/
        void delay_msec(unsigned int count)
{
        unsigned int i;
        TMOD = 0X01;
        //Configure Timer/Counter0 as timer for mode1(16-bit count)
        TR0  = 1;
        //Start Timer0
                for(i = 0; i < count; i++)//Loop as long as required delay
is attained
        {
                TH0 = 0XF8;
        //Assign value to Timer0 register to generate 1 msec delay
                TL0 = 0XCC;
                while(!TF0);
        //Loop here until Timer0 overflow flag gets set
                TF0 = 0;
        //Clear Timer0 overflow flag to check next overflow
        }
        TR0 = 0;
        //Stop Timer0
}
```

3. Write an 8051 C Program to rotate the D.C motor in Clockwise & anticlockwise direction.

```
 #include<reg52.h>
#define DATA_BUS       P2
#define ADD_BUS        P0
void main(void)
{
        void delay_msec(unsigned int count);
        while(1)
        {
                DATA_BUS = 1;
                ADD_BUS  = 0X07;
                ADD_BUS  = 0XFF;
                delay_msec(90);
                DATA_BUS = 0x00;
                ADD_BUS  = 0X07;
                ADD_BUS  = 0XFF;
                delay_msec(10);
        }
}

void delay_msec(unsigned int count)
{
        unsigned int i;
        TMOD = 0X01;
        TR0 = 1;
        for(i = 0; i < count; i++)
        {
                TH0 = 0XF8;
                TL0 = 0XCD;
                while(!TF0);
                TF0 = 0;
        }
}
```

4. Write an ALP to generate a Triangular wave using DAC.

```
#include"reg52.h"
#include"intrins.h"

#define  ADD_BUS P0                  //ADDRESS BUS
#define  DAC_ADD 0X21                //DAC ADDRESS
sbit P20=P2^0;
sbit P21     =P2^1;
sbit P22=P2^2;
sbit P23=P2^3;
sbit P24=P2^4;
sbit P25=P2^5;
sbit P26=P2^6;
sbit P27=P2^7;
sbit P30=P3^0;
sbit P31=P3^1;
sbit P32=P3^2;
sbit P33=P3^3;
sbit P34=P3^4;
sbit P35=P3^5;
sbit P36=P3^6;
sbit P37=P3^7;
      void main(void)
{
      unsigned char count = 0;   //DELCARE COUNT VARIABLE AS
                                   UNSIGNED CHARACTER TYPE

      bit flag = 1;                //FLAG IS NEED TO SELECT POSTIVE
                                  AND NEGATIVE OF HALF CYCLE

      ADD_BUS  = DAC_ADD;       //ENABLE THE DAC TO ASSIGN
              DIGITAL DATA WHICH IS PLACE ON THE DATA BUS

         while(1)
      {
            P3 = count;                //ASSIGN COUNT VALUE TO P3
            P20 = P37;                 //ASSIGN P3.7 VALUE TO P2.0
                        BECAUSE P2.0 IS HAVING HIGHEST WEIGHT

            P21 = P36;                 //ASSIGN P3.6 VALUE TO P2.1
            P22 = P35;                 //ASSIGN P3.5 VALUE TO P2.2
```

```
                P23 = P34;              //ASSIGN P3.4 VALUE TO P2.3
                P24 = P33;              //ASSIGN P3.3 VALUE TO P2.4
                P25 = P32;              //ASSIGN P3.2 VALUE TO P2.5
                P26 = P31;              //ASSIGN P3.1 VALUE TO P2.6
                P27 = P30;              //ASSIGN P3.0 VALUE TO P2.7
                    if(flag)            //IF FLAG IS SET INDCATES
                  STILL COUNT VALUE IS NOT INCREASED TO FFH
                {
                    count++;    //SO INCREASE THE COUNT VALUE
                if(count==0xff)  // IF COUNT VALUE IS EQUAL TO FFH
                    {
                        flag = 0;               //REESET THE FLAG
                    }
                }
                    else
                {
                    count--;        //IF FLAG IS RESET INDICATES
                        COUNT VALUE IS NOT DECREASED TO 00H
                if(count == 0)      //IF COUNT VALUE IS EQUAL TO 00H
                    {
                        flag = 1;               //SET THE FLAG
                    }
                }
        }
}
```

Output :

5. Write an ALP to generate a Square wave using DAC.

```
#include"reg52.h"

#define  ADD_BUS P0                 //ADDRESS BUS
#define  DATA_BUS P2                //DATA BUS
#define  DAC_ADD 0X21               //DAC ADDRESS
#define  DISABLE 0XFF               // ADDRESS VALUE DISABLE
                                    ALL PERIPERHAL DEVICES
      void main(void)
{
      void delay_msec(unsigned int count);

      ADD_BUS  = DAC_ADD;       //ENABLE THE DAC TO ASSIGN
              DIGITAL DATA WHICH IS PLACE ON THE DATA BUS
            while(1)
      {
            DATA_BUS = 0X00;              //PLACE 00H ON THE DATA
                              BUS TO GENERATE ACTIVE LOW PULSE
            delay_msec(5);                //APPLY ONE MILLISECOND
                             DELAY AS ACTIVE LOW PULSE WIDTH
            DATA_BUS = 0XFF;      //PLACE FFH ON THE DATA BUS
                                TO GENERATE ACTIVE HIGH PULSE
            delay_msec(5);                //APPLY ONE MILLISECOND
                           DELAY AS ACTIVE HIGH PULSE WIDTH
      }
}

/*---------------------------------------------------------------------------
FUNCTION NAME      : TIMER0 AS DELAY GENERATOR

DESCRIBITION  :       IN THIS FUNCTION, TIMER/COUNTER0 IS
CONFIGURED AS DELAY GENERATOR WITH
                                    RESOLUTION OF 1 msec.
                                    NOTE: CLOCK/CYCLE = 6
                                    TH0TL0 = 65536 - ((11.0592 X 10^6)
X DELAY RESOLUTION)/(CLOCK/CYCLE)


AND CLOCK/CYBLE=6
                                    THEN TH0TL0 = 65536 - (11.0592 X
10^6) X 1X10^3)/6       = F8CCH
```

```
-----------------------------------------------------------------------------------*/
    void delay_msec(unsigned int count)
{
    unsigned int i;
    TMOD = 0X01;
    //CONFIGURE TIMER/COUNTER0 AS TIMER FOR MODE1
                        (16-BIT COUNT)
    TR0  = 1;
    //START TIMER0
    for(i = 0; i < count; i++)  //LOOP AS LONG AS REQUIRED
                                DELAY IS ATTAINED
    {
        TH0 = 0XF8;
    //ASSIGN VALUE TO TIMER0 REGISTER TO GENERATE 1
                                                mSEC DELAY
        TL0 = 0XCC;
        while(!TF0);
    //LOOP HERE UNTIL TIMER0 OVERFLOW FLAG GETS SET
        TF0 = 0;
    //CLEAR TIMER0 OVERFLOW FLAG TO CHECK NEXT
                                            OVERFLOW
    }
    TR0 = 0;
    //STOP TIMER0
}
```

Output:

Beyond Syllabus

1. **Write an C program to generate a up Saw tooth wave using DAC.**

```
#include"reg51.h"
#define ADD_BUS P0
#define DAC_ADD

Sbit P20=P2^0;
Sbit P21=P2^1;
Sbit P22=P2^2;
Sbit P23=P2^3;
Sbit P24=P2^4;
Sbit P25=P2^5;
Sbit P26=P2^6;
Sbit P27=P2^7;
Sbit P30=P3^0;
Sbit P31=P3^1;
Sbit P32=P3^2;
Sbit P33=P3^3;
Sbit P34=P3^4;
Sbit P35=P3^5;
Sbit P36=P3^6;
Sbit P37=P3^7;
void main(void)
{
  unsigned char count  =  0;
  ADD_BUS  =  DAC_ADD;
    while(1)
        {
          P3 = count;
          P20 = P37;
          P21 = P36;
          P22 = P35;
          P23 = P34;
          P24 = P33;
          P25 = P32;
          P26 = P31;
          P27 = P30;
          count++;
          }
        }
```

2. Write an C program to generate a Down Saw tooth wave using DAC.

```
#define ADD_BUS P0
#define DAC_ADD
Sbit P20=P2^0;
Sbit P21=P2^1;
Sbit P22=P2^2;
Sbit P23=P2^3;
Sbit P24=P2^4;
Sbit P25=P2^5;
Sbit P26=P2^6;
Sbit P27=P2^7;
Sbit P30=P3^0;
Sbit P31=P3^1;
Sbit P32=P3^2;
Sbit P33=P3^3;
Sbit P34=P3^4;
Sbit P35=P3^5;
Sbit P36=P3^6;
Sbit P37=P3^7;
void main(void)
{
  unsigned char count  =  0;
  ADD_BUS  =  DAC_ADD;
    while(1)
        {
          P3 = count;
          P20 = P37;
          P21 = P36;
          P22 = P35;
          P23 = P34;
          P24 = P33;
          P25 = P32;
          P26 = P31;
          P27 = P30;
          Count--;
          }
        }
```

VIVA VOCE Questions

- What is 8051 Microcontroller ?
- What are registers in Microcontroller ?
- List Interrupts available in 8051 Microcontroller.
- What is stack pointer in 8051 Microcontroller?
- List some features of 8051 Microcontroller.
- What is an Interrupt service routine in Microcontroller?
- What is an interrupt?
- Compare microprocessor and controller
- Compare risc and cisc
- What is the difference between timer and counter of microcontroller?
- Explain Serial communication flags and registers used in microcontroller.
- Explain addressing modes used in microcontroller programming
- What Is The Difference Between Harvard Architecture And Von Neumann Architecture?
- What Is The Width Of Data Bus?
- What Location Code Memory Space And Data Memory Space Begins
- How Much On Chip Ram Is Available?
- How Much Total External Data Memory That Can Be Interfaced To The 8051?
- What Is Special Function Registers (sfr)?
- What Are The Four Distinct Types Of Memory In 8051?
- Explain assembler directives
- Which Bit Of The Flag Register Is Set When Output Overflows To The Sign Bit?
- Explain branching instructions
- Explain conditional and unconditional jmp instructions
- Which 2 Ports Combine To Form The 16 Bit Address For External Memory Access?(Port0 and port2)