INSTITUTE OF AERONAUTICAL ENGINEERING
(Autonomous)
Dundigal - 500 043, Hyderabad, Telangana

COURSE CONTENT

MICROPROCESSORS AND MICROCONTROLLERS LABORATORY								
V Semester: ECE / EEE								
Course Code	Category	Hours / Week			Credits	Maximum Marks		
AECC31	Core	L	T	P	C	CIA	SEE	Total
		0	0	3	1.5	30	70	100
Contact Classes: Nil	Tutorial Classes: Nil	Practical Classes: 36				Total Classes: 36		
Prerequisite: Digital System Design								

I. COURSE OVERVIEW:
This laboratory course will facilitate the students to program 8086 microprocessor and 8051 microcontrollers. Win862 software will be used for writing and debugging assembly language programs. The course includes performing arithmetic and logical operations, string manipulations, code conversions and interfacing of I/O devices to processor/controller. The hands-on experience acquired by the student's during the course makes them to carry out processor/controller-based projects and extend their knowledge on the latest trends and technologies in the field of embedded system.

II.COURSE OBJECTIVES:
The students will try to learn:

I. Assembly language programming skills ranging from simple arithmetic operations to interfacing real time systems.
II. The usage of software tools to design, debug and test microprocessor/microcontroller based projects using assembly language programming.
III. The design of microcomputer and microcontroller based real-time applications in the fields of communication systems, home based automation systems, automobiles and unmanned applications

III. COURSE OUTCOMES:
At the end of the course students will be able to:

CO1 Make use of emulators and assemblers for writing, compiling, and running an assembly language programs on training boards.
CO2 Develop Assembly language programs for accomplishing code conversions, string manipulations and sorting of numbers.
CO3 Choose serial or parallel communication for transmitting the data between microprocessor or microcontroller and peripherals.
CO4 Utilize Analog to Digital and Digital to Analog converters with processor or controller for data conversion.
CO5 Select suitable registers of microcontroller and write assembly language program to verify timer or counter operations.
CO6 Build an interface between processor or controller and peripherals to provide solutions to the real-world problems

IV. COURSE CONTENT:

Exercises for Microprocessors and Microcontrollers Laboratory

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

1. Getting Started Exercises

1.0 Introduction:

- Install Win862 assembler software.
- Interface desktop to 8086 microprocessor kit through RS232 serial communication cable.
- Then write the assembly code using instruction set and logical function of program
- Decode and run the program execution
- Check the result in register pallet or memory location.

1.1 Arithmetic Operations

- The arithmetic instructions include addition, subtraction, multiplication, division, comparison, negation, increment, and decrement.
- Write a assembly level language program using arithmetic instructions to find the addition, subtraction and multiplication between the unsigned numbers.

Preparation: 8086 Architecture, Instruction set of 8086 and Register-Memory Organization

Input: numbers = [08, 02]

Output:

i. For addition : 0A

ii. For subtraction : 06

iii. For multiplication : 0010

```
find_addition of unsigned numbers(8 bit):
# Write code here
     …
     …

find_subtraction of unsigned numbers (8 bit):
# Write code here
     …
     …
find_multiplication of unsigned numbers (8 bit):
# Write code here
     …
     …
```

1.2 Logical Operations

- The logic instructions include AND, OR, Exclusive-OR, NOT, shifts, rotates, and the logical compare (TEST).
- Using logical instructions to find the logical operations like AND,OR,EX-OR etc between the unsigned numbers.

Preparation: 8086 Architecture, Instruction set of 8086 and Register-Memory Organization

Input: 05 and 07

Output:

i. For AND logic : 0A

ii. For OR logic : 06

iii. For EX-OR logic : 0010

```
find_AND logic of unsigned numbers(8 bit):
# Write code here
     …

find_ OR logic of unsigned numbers (8 bit):
# Write code here
     …

find_ EX-OR logic of unsigned numbers (8 bit):
# Write code here
```

1.3 Shift Operations

Instructions to perform shift operations

- **SHL/SAL** – Used to shift bits of a byte/word towards left and put zero(S) in LSBs.
- **SHR** – Used to shift bits of a byte/word towards the right and put zero(S) in MSBs.
- **SAR** – Used to shift bits of a byte/word towards the right and copy the old MSB into the new MSB.

- **Syntax:** SHL/SAR/SHR Register, Bits to be shifted

Preparation: 8086 Architecture, Instruction set of 8086 and Register-Memory Organization

Input: = 12(0001 0010)
No of shifts = 04

Output:

- For **SHL/SAL** – **20** (0010 0000)
- For **SHR** – **01** (0000 0001)
- For **SAR** – **01** (0000 0001)

```
# Program to perform shift operations for given numbers

SHL/SAL logic:
       # Write code here
…
SAR logic:
```

```
        # Write code here
 …
SHR  logic:
        # Write code here
```

1.4 Rotate Operations

Write a assembly level language program using instructions sets to find the rotation operations between the unsigned numbers.

Instructions to perform rotate operations

- **ROL** – Used to rotate bits of byte/word towards the left, i.e. MSB to LSB and to Carry Flag [CF].
- **ROR** – Used to rotate bits of byte/word towards the right, i.e. LSB to MSB and to Carry Flag [CF].
- **RCR** – Used to rotate bits of byte/word towards the right, i.e. LSB to CF and CF to MSB.
- **RCL** – Used to rotate bits of byte/word towards the left, i.e. MSB to CF and CF to LSB.

- **Syntax:** ROL/ROR/RCR/RCL Register, Bits to be shifted

Preparation: 8086 Architecture, Instruction set of 8086 and Register-Memory Organization

Input: = 12(0001 0010)
No of shifts = 04

Output:

- For **ROL** –
- For **ROR** –
- For **RCR** –
- For **RCL** –

```
# Program to perform rotate operations for given numbers

ROL logic:
        # Write code here
…
ROR  logic:
        # Write code here
 …
RCR  logic:
        # Write code here
…
RCL  logic:
        # Write code here
```

1.5 Multibyte Addition

A multibyte character can hold code-point values greater than 255. One multibyte character can range 2 - 4 bytes in length. Asian code sets are multibyte code sets; they contain both single-byte and multibyte characters.

The multi-byte addition program adds only in sets of 8-bits. The LSD of the two numbers is added first. Now, the next set of 8-bits is added, taking into consideration the status of carry due to the previous addition.

Write an 8086 Assembly Language program to Add multi byte numbers and store the result in memory

Preparation: 8086 Architecture, Instruction set of 8086 and Register-Memory Organization

Input and Output: count of inputs=03

Input		**Output**	
Memory	**Data**	**Memory**	**Data**
2000		4000	
2001		4001	
2002		4002	
3000			
3001			
3002			

```
# Multibyte addition program:
BACK: MOV AL, [SI]
ADD AL, [BX]
MOV [DI], AL
INC SI
INC BX
INC DI
DEC CX
JNZ BACK
INT 03
```

1.6 Multibyte Subtraction:

A multibyte character can hold code-point values greater than 255. One multibyte character can range 2 - 4 bytes in length. Asian code sets are multibyte code sets; they contain both single-byte and multibyte characters.

The multi-byte addition program adds only in sets of 8-bits. The LSD of the two numbers is added first. Now, the next set of 8-bits is added, taking into consideration the status of carry due to the previous addition.

Write an 8086 Assembly Language program to Add multi byte numbers and store the result in memory

Preparation: 8086 Architecture, Instruction set of 8086 and Register-Memory Organization

Input and Output: count of inputs=03

Input		Output	
Memory	**Data**	**Memory**	**Data**
2000		4000	
2001		4001	
2002		4002	
3000			
3001			
3002			

```
# Multibyte subtraction program:
BACK: MOV AL, [SI] SUB AL, [BX]
MOV [DI], AL
INC SI
INC BX
INC DI
DEC CX
JNZ BACK
INT 03
```

Try: Write an assembly language program for performing arithmetic and logical operations of two 16-bit numbers

A 16-bit number is a number with 16 digits, where the largest number is 65,535. A 16-bit register can store a positive number between 0 and 216 − 1, or 65,535.

Preparation: 8086 Architecture, Instruction set of 8086 and Register-Memory Organization

Write an assembly language program for performing arithmetic operations of two 16-bit numbers

Input: AX =4343

BX = 1111

Output: 5454

```
For_ arithmetic operations:
# Write code here
--
--
ADD AX, BX
--
# Write code here
--
```

```
--
--

For_ Logical operations:
# Write code here
--
--
--
```

2.Exercises on matrices, code conversions

2.1 Matrix Addition

Given two **N** x **M matrices**. Find a **N** x **M** matrix as the sum of given matrices each value at the sum of values of corresponding elements of the given two matrices.

Iterate over every cell of matrix (i, j), add the corresponding values of the two matrices and store in a single matrix i.e. the resultant matrix.

Input and Output:

Input Matrix 1:	**Input Matrix 2:**	**Output:**
11 22 33	66 55 44	77 77 77
44 55 66	66 55 44	AA AA AA
77 88 99	66 55 44	DD DD DD

```
# Implementation of above approach

To perform the addition of two 3x3 matrices using Assembly language for 8086 microprocessors.
```

```
# Write code here
…
```

2.2 Matrix Multiplication

Given two **N** x **M matrices**. Find a **N** x **M** matrix as the product of given matrices each value at the product of values of corresponding elements of the given two matrices. *Iterate over every cell of matrix (i, j), multiply the corresponding values of the two matrices and store in a single matrix i.e. the resultant matrix.*

Input and Output:

Matrix 1:	**Matrix 2:**	**Output:**
11 22 33	66 55 44	1E 1E 1E
44 55 66	66 55 44	3C 3C 3C
77 88 99	66 55 44	5A 5A 5A

```
# Implementation of above approach

To perform the multiplication of two 3x3 matrices using Assembly language for 8086 microprocessor.
# Write code here
…
```

2.3 Packed BCD number to Unpacked BCD number

Write a program to convert the codes from packed BCD to unpacked BCD number

They can only understand the data in the form of 0's and 1's. Some of them are the Binary number system, Octal number system, Hexadecimal number system, etc. To make the text understandable by computers ASCII codes are used.

Internal converters are used for converting data from one format to another. In this the code conversion involves operations like Packed BCD to Unpacked BCD, BCD to ASCII, Hexadecimal number to ASCII number. The microprocessor understands the binary/hex number system.

Preparation: 8086 Architecture, Instruction set of 8086 and Register-Memory Organization

Instructions to perform shift operations

- **SHL/SAL** – Used to shift bits of a byte/word towards left and put zero(S) in LSBs.
- **SHR** – Used to shift bits of a byte/word towards the right and put zero(S) in MSBs.
- **SAR** – Used to shift bits of a byte/word towards the right and copy the old MSB into the new MSB.

- **Syntax:** SHL/SAR/SHR Register, Bits to be shifted

Preparation: 8086 Architecture, Instruction set of 8086 and Register-Memory Organization

Input		Output	
2000	72	AH	07
		AL	02

```
# Packed BCD to Unpacked BCD
MOV AX,0000
MOV AL,72
MOV AH,AL
AND AL,0F
MOV CL,04
SHR AH,CL
INT 03
```

2.4 BCD to ASCII conversion

Given an instruction set of 8086 microprocessor, write a program to convert a number from BCD to ASCII

Input:

DATA: 98H in memory location

Output:

DATA: 38H in memory location 1 and
39H in memory location 2

```
# BCD to ASCII conversion
MOV AL,98
MOV AH,AL
AND AL,0F
MOV CL,04
SHR AH,CL
OR AX,3030
INT 03
```

2.5 Covert Hexadecimal number to ASCII number

Given the Hexadecimal value string as input, the task is to convert the given hexadecimal value string into their corresponding ASCII format string.

Example:
Input: 2050 E4 (Hex data)
Output:
2051 34 (ASCII code for 4)
2052 45 (ASCII code for E)

The "Hexadecimal" or simply "Hex" numbering system uses the Base of 16 system. Being a Base-16 system, there are 16 possible digit symbols. Hexadecimal number uses 16 symbols {0, 1, 2, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F} to represent all numbers. Here, (A, B, C, D, E, F) represents (10, 11, 12, 13, 14, 15).

ASCII stands for American Standard Code for Information Interchange. ASCII is a standard that assigns letters, numbers, and other characters within the 256 slots available in the 8-bit code.

```
# Hexadecimal number to ASCII number

MOV SI,2000
MOV DI,3000
MOV CX,0003
UP MOV AL,[SI]
CMP AL,0A
JC FWD
ADD AL,07
FWD OR AL,30
MOV [DI], AL
INC SI
INC DI
DEC CX
JNZ UP
INT 03
```

2.6 ASCII to BCD conversion

This program can change ASCII value of a number to its BCD (Decimal) form. The ASCII values of the numbers are like below:

ASCII (in Hex)	30	31	32	33	34	35	36	37	38	39
BCD	00	01	02	03	04	05	06	07	08	09

From this table we can easily find that the last nibble of the ASCII value is actually the BCD equivalent. So to take the last nibble we have masked the upper nibble, and take the lower nibble as result.

Input: 39 and 38
Output: 89

```
# Convert ASCII to BCD
# Class to convert the expression
MOV AL,39
MOV AH,38
AND AL,0F
SHL AH,04
OR AL,AH
INT 03
```

3. Exercises on Sorting and string manipulation

3.1 Sort the number in ascending order

In this sorting technique there will be **n** passes for **n** different numbers. In i^{th} pass the i^{th} largest element will be placed at the end. This is comparison based sort. We taking two consecutive numbers, compare them, and then swap them if the numbers are not in correct order. The following diagram is showing how the sorting is working.

Preparation: 8086 Architecture, Instruction set of 8086 and Register-Memory Organization

Input: 06, CF, 2C, 51, 45, 24, 3E
Output: 06, 24, 2C, 3E, 45, 51, CF

```
# Sorting ascending order
MOV AX,0000
MOV CH,0004
DEC CH
UP1 : MOV CL,CH
MOV SI,2000
UP:  MOV AL,[SI]
INC SI
CMP AL,[SI]
JC DOWN
XCHG AL,[SI]
DEC SI
MOV [SI],AL
INC SI
DOWN : DEC CL
```

```
JNZ UP
DEC CH
JNZ UP1
INT 03
```

3.2 Sort the number in descending order

Descending order means the largest or last in the order will appear at the top of the list:

- For numbers or amounts, the sort is largest to smallest. Higher numbers or amounts will be at the top of the list.
- For letters/words, the sort is alphabetical from Z to A.
- For data with numbers and letters/words, such as address lines, the sort is most likely alphanumeric meaning Z to A is sorted first then followed by 9-0.

This is comparison based sort. We taking two consecutive numbers, compare them, and then swap them if the numbers are not in correct order

Input: arr = [11, 12, 22, 25, 64]

Output: arr = [64, 25, 12, 22, 11]

```
# Implementation of selection sort
# Implementation of Bubble Sort

def descending order Sort(arr):
       #write code here
       …

MOV AX,0000
MOV CH,0004
DEC CH
UP1 : MOV CL,CH
MOV SI,2000
UP:  MOV AL,[SI]
INC SI
CMP AL,[SI]
JNC DOWN
XCHG AL,[SI]
DEC SI
MOV [SI],AL
INC SI
DOWN : DEC CL
JNZ UP
DEC CH
JNZ UP1
INT 03
```

3.3 Insert a byte

String is a group of bytes/words and their memory is always allocated in a sequential order. String is either referred as byte string or word string. Here we will see some instructions which are used to manipulate the string related operations

Preparation: 8086 Architecture, Instruction set of 8086 and Register-Memory Organization

Inputs	Outputs
2000-01	3000-01
2001-02	3001-02
2002-03	3002-03
2003-05	3003-04
2004-06	3004-05
5000-03	3005-06
7000-04	

```
# Program for insert a byte in a string

MOV SI,2000H
MOV DI,3000H
MOV BX,5000H
MOV CX,0005H
CLD
L1:MOV AL,[SI]
CMP AL,[BX]
JZ L2
MOVSB
LOOP L1
JMP L3
L2:    MOVSB
MOV BX,7000H
MOV AL,[BX]
MOV [DI],AL
DEC CX
INC DI
REP MOVSB
L3:INT 3
```

3.4 Delete a byte

CLD instruction is used to clear the directional flag, i.e., DF=0. Now, value of SI and DI will be increased.
SI=SI+1

DI=DI+1

REP instruction is used to repeat the step until the value of CX is not equal to zero and the value of CX is decremented by one at every step, i.e.,
CX=CX-1

MOVSB instruction is used to transfer bytes only from source memory location (MADS) to destination memory location (MAES).

MADS-->MAES
Where, MADS=DS*10+SI
MAES=ES*10+DI
Here, value of SI and DI is updated automatically.
if DF=0, SI=SI+1 and DI=DI+1

Inputs	Outputs
2000-01	3000-01
2001-02	3001-02
2002-03	3002-04
2003-04	3003-05
2004-05	
5000-03	

```
# Delete a byte in a given string

MOV SI,2000H
MOV DI,3000H
MOV BX,5000H
MOV CX,0005H
CLD
L1:MOV AL,[SI]
CMP AL,[BX]
JZ L2
MOVSB
LOOP L1
JMP L3
L2:INC SI
DEC CX
REP MOVSB
L3:INT 03H
```

3.5 Searching of a number in a string

Search a number in a string of 5 bytes, store the offset where the element is found and the number of iterations used to find the number

Inputs	Outputs
2000-01	**3000-02**
2001-02	**AH-01**
2002-03	
2003-04	
5000-02	

```
# Searching of  a number
MOV CX, 0004
MOV AX,0000
```

```
MOV SI,2000
MOV BX,3000
UP:MOV AL,[SI]
CMP AL,[BX]
JZ DOWN
INC SI
DEC CL
JNZ UP
MOV AH,00
JMP L3
DOWN:  DEC CL
MOV AH,01
MOV [DI], AH
L3:INT 3
```

3.6 Transfer a block of data

program to transfer a block of 4 bytes, starting source address and transfer the block at destination address by using string instructions.

MOVSB instruction is used to transfer bytes only from source memory location (MADS) to destination memory location (MAES).
MADS-->MAES

Where, MADS=DS*10+SI

MAES=ES*10+DI

Here, value of SI and DI is updated automatically.

Input		Output	
2000	01	2008	01
2001	02	2009	02
2002	03	200A	03
2003	04	200B	04
2004	05	200C	05

```
# program to transfer the block of data
MOV DI, 2008H
MOV CX, 0005H
REP MOVSB
INT 03H
```

Try: Reverse of a given string

Program to reverse a number of 4 bytes, Reverse the contents of a register lower by executing RLC instruction 4 times, Reverse the contents of register higher by executing RLC instruction 4 times, Store the content of register in memory location.

Inputs		outputs	
2000	01	2008	05
2001	02	2009	04
2002	03	200A	03
2003	04	200B	02
2004	05	200C	01

```
# Reverse of a given string
MOV SI, 2000H
MOV DI, 2008H
MOV CX, 0008H
ADD SI, 07H
UP:MOV AL,[SI]
MOV [DI], AL
DEC SI
INC DI
DEC CX
JNZ UP
INT 03H
```

Try: Search for given string

Search a number in a string of 5 bytes, store the offset where the element is found and the number of iterations used to find the number.

Write a program to Search a number in a string to find the number

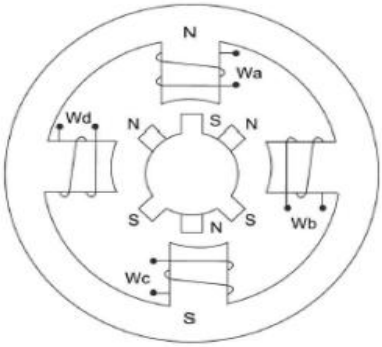
```
# Search for a number in a given string

MOV AX, 2000
MOV ES, AX
MOV DI, 600
MOV AX, 25
MOV CX, 0005
MOV BX, CX
CLD
REPNE SCAS B
DEC DI
MOV DX, DI
SUB BX, CX
DEC BX
INT 03
```

4. Exercises on Interfacing of stepper motor to 8086

4.1 clockwise rotation of stepper motor

Interfacing of stepper motor to 8086 microprocessor and rotates it in clockwise and anti-clock wise direction. A stepper motor is a type of DC motor that rotates in steps.



Preparation: 8086 Architecture, Instruction set of 8086, 8255 PPI, Control word register

Motion	step	A	B	C	D
Clock Wise Direction	1	1	0	0	0
	2	0	1	0	0
	3	0	0	1	0
	4	0	0	0	1
	5	1	0	0	0
Anti clock wise Direction	1	1	0	0	0
	2	0	0	0	1
	3	0	0	1	0
	4	0	1	0	0
	5	1	0	0	0

Write a ALP program to rotate the stepper motor in clockwise and anti clock wise direction using 8086

```
# stepper motor rotation in clockwise direction

MOV AL,80
MOV DX,0FFE6
OUT DX
MOV BX,1770
MOV AL,33
MOV DX,0FFE0
BACK OUT DX
MOV CX,1262
SELF LOOP SELF
ROR AL,1
DEC BX
JNZ BACK
INT 03
```

4.2 Anticlockwise rotation of stepper motor

Interfacing of stepper motor to 8086 microprocessor and rotates it in anti clock wise direction.

```
stepper motor rotation in Anticlockwise direction

MOV AL,80
MOV DX,0FFE6
OUT DX
MOV BX,1770
MOV AL,33
```

```
MOV DX,0FFE0
BACK OUT DX
MOV CX,1262
SELF LOOP SELF
ROL AL,1
DEC BX
JNZ BACK
INT 03
```

4.3 Analog to Digital conversion

Many events monitored and controlled by the microprocessor are analog events. The ADC and DAC devices are used to interface the microprocessor to the analog world. Analog-to-Digital Converters (ADC's) convert analog signals to digital data.

Digital-to-Analog converters (DAC's) convert digital data to analog signals. They are a common peripheral used with microprocessors for applications such as controlling analog circuitry, audio and video generation, radio signal generation, etc.

Preparation: 8086 Architecture, Instruction set of 8086, 8255 PPI, Control word register

Analog I/P selected	Address lines		
	C	B	A
I/P 0	0	0	0
I/P 1	0	0	1
I/P 2	0	1	0
I/P 3	0	1	1
I/P 4	1	0	0
I/P 5	1	0	1
I/P 6	1	1	0
I/P 7	1	1	1

```
# Analog to digital converter
MOV AL, 98H
MOV DX, 0FFE6
OUT DX,AL
MOV AL, 01H
OUT DX,AL
MOV AL, 00H
OUT DX,AL
MOV AL, 02H
MOV DX, 0FFE2H
OUT DX,AL
MOV DX, 0FFE4H
IN AL,DX
ROR AL, 1H
JNC BACK
MOV DX, 0FFE0H
BACK: IN AL,DX
MOV DI, 2000H
MOV [DI], AL
INT 03H
```

4.4 Digital to Analog converter-Square wave generator

The digital to analog converters convert binary numbers into their analog equivalent voltages. The DAC find applications in areas like digitally controlled gains, motor speed controls, programmable gain amplifiers, etc.

```
# Square wave generator.
MOV AL,80
MOV DX,0FFE6
OUT DX
MOV DX,0FFE0
BACK: MOV AL,00
OUT DX
MOV CX,0147
SELF1: LOOP SELF1
MOV AL,0FF
 OUT DX
MOV CX,0147
SELF2 : LOOP SELF2
JMP BACK
```

4.5 Digital to Analog converter-Triangular wave generator

The digital to analog converters convert binary numbers into their analog equivalent voltages. The DAC find applications in areas like digitally controlled gains, motor speed controls, programmable gain amplifiers, etc.

```
# Triangular wave generator.
MOV AL,80
MOV DX,0FFE6
OUT DX
MOV AL,00
L3 MOV DX,0FFE2
L1 OUT DX
INC AL
CMP AL,0FF
JB L1
L2 OUT DX
DEC AL
CMP AL,00
JNBE L2
JMP L3
```

Try: Digital to Analog converter-Sawtooth wave generator

The digital to analog converters convert binary numbers into their analog equivalent voltages. The DAC find applications in areas like digitally controlled gains, motor speed controls, programmable gain amplifiers, etc.

```
# Saw tooth wave generator.
Write code
….
….
```

Try : Rotation of stepper motor with 60 rotations with 8MHz

Interfacing of stepper motor to 8086 microprocessor and rotates it in anti clock wise direction.

```
stepper motor rotation in Anticlockwise direction

MOV AL,80
MOV DX,0FFE6
OUT DX
MOV BX,1770
MOV AL,33
MOV DX,0FFE0
BACK OUT DX
MOV CX,1262
SELF LOOP SELF
ROL AL,1
DEC BX
JNZ BACK
INT 03
```

5. Exercises on programmable peripherals

5.1 Keyboard operation

The keyboard consists of maximum 64 keys, which are interfaced with the CPU by using the key-codes. These key-codes are de-bounced and stored in an 8-byte FIFORAM, which can be accessed by the CPU.

Preparation: 8086 Architecture, Instruction set of 8086, 8255 PPI, Control word register

```
# Keyboard operation

# Write code here
        …

# Driver Code
```

5.2 Matrix Keyboard operation

In a matrix keyboard there are keys which are arranged in the form of a matrix which consists of several rows and columns.

connect a key at the intersection of all rows and columns. Hence there is a total of 4 × 4 = 16 keys in the given matrix. The lines of the columns get connected to Gnd through pull-down resistors.

Write an ALP program to execute 3x3 matrix keyboard operation

Preparation: 8086 Architecture, Instruction set of 8086, 8255 PPI, Control word register

```
# Keyboard operation

# Write code here
        …

# Driver Code
```

5.3 Serial Communication

In serial communication we send only q bit at a time, one after the other. So 8 bits need 8 times the time required, compared to the previous case. The advantage is only one physical wire is required for transmission.

Program to receive bytes of data serially, and put them in P2, set the baud rate at 9600, 8-bit data, and 1 stop bit:

Preparation: 8086 Architecture, Instruction set of 8086, 8255 PPI, Control word register

```
# Serial Communication
# Write code here
MOV TMOD, #20H
MOV TH1, #-3
MOV SCON, #50H
SETB TR1
HERE: JNB RI, HERE
MOV A, SBUF
MOV P2, A
CLR RI
SJMP HERE
```

5.4 Parallel Communication

Data is to be sent from the source to the destination, and it is necessary for the source and destination formats to be similar for compatibility between them. In parallel communication all the bits are sent and received together.

```
# Parallel Communication

# Write code here
        …

# Driver Code
```

5.6 Interfacing traffic light controller

Introduction Traffic Light interface module is designed to simulate the function of four way traffic light controller. Combination of Red, Yellow, Green LED's are provided to indicate Halt, Wait, Go. Combination of Red and Green LED's are provided for pedestrian crossing.

Write an ALP program to generate a signal using 8086 to interface to traffic light controller interface board

Preparation: 8086 Architecture, Instruction set of 8086, 8255 PPI, Control word register

```
# Interfacing traffic light controller

# Write code here
        …

# Driver Code
```

6. Exercises on Arithmetic and Logical operations using 8051 Microcontroller

6.1 Arithmetic operations

The assembly language programs for performing arithmetic and logical operations are composed by using mnemonics, various addressing modes, instructions and registers of microcontroller. The 8051 microcontroller is used to execute the instructions of assembly language program one by one.

Using arithmetic instructions to find the addition, subtraction and multiplication between the unsigned numbers.

In this case, arithmetic instructions via assembly programming language to find the arithmetic operations for provided list of numbers. The output will be display in hexadecimal format.

Write a alp program arithmetic and logical operations using 8051 Microcontroller

Preparation: 8051 Architecture, Instruction set of 8051 and Register-Memory Organization

Input: numbers = [08, 02]

Output:

iv. For addition : 0A

v. For subtraction : 06

vi. For multiplication : 0010

```
find_addition of unsigned numbers(8 bit):
# Write code here
     …

find_subtraction of unsigned numbers (8 bit):
# Write code here
     …

find_multiplication of unsigned numbers (8 bit):
# Write code here
```

6.2 Logical Operations

The logic instructions include AND, OR, Exclusive-OR, NOT, shifts, rotates, and the logical compare (TEST). Using logical instructions to find the logical operations like AND,OR,EX-OR etc between the unsigned numbers.

In this case, logical instructions via assembly programming language to find the logical operations for provided list of numbers. The output will be display in hexadecimal format.

Preparation: 8051 Architecture, Instruction set of 8051 and Register-Memory Organization

Input: 05 and 07

Output:

iv. For AND logic : 0A

v. For OR logic : 06

vi. For EX-OR logic : 0010

```
find_AND logic of unsigned numbers(8 bit):
# Write code here
     …

find_ OR logic of unsigned numbers (8 bit):
# Write code here
     …

find_ EX-OR logic of unsigned numbers (8 bit):
# Write code here
```

Try: Store the data from various memory location

Using 8051 microcontroller 0 instruction sets to store the data in various memory locations for given array of numbers

```
     …

find_ OR logic of unsigned numbers (8 bit):
# Write code here
     …
find_ EX-OR logic of unsigned numbers (8 bit):
# Write code here
```

7. Exercises on Timer/Counter

7.1 Asymmetric square wave

A timer is a specialized type of clock which is used to measure time intervals. A timer that counts from zero upwards for measuring time elapsed is often called a stopwatch. It is a device that counts down from a specified time interval and used to generate a time delay, for example, an hourglass is a timer.

An asymmetric square wave is a square wave with a duty cycle that is not 50%. A square wave is a wave that is square and has a 50% duty cycle..

Generate an asymmetric square wave of 120Hz and having a duty cycle of 25% using the Timer0 module.

```
# Write code here

#include<lpc214x.h>
//124373 * 0.25 = 31093 = 7975H
//124373 * 0.75 = 93280; 93280/2 = 46640 = B630
voidon_delay(void){
T0MR0=0x7974;
T0PR=0;
T0TCR=1;
while(T0TC!=T0MR0);
T0TCR=2;
T0TC=0;
}
voidoff_delay(void){
T0MR0=0xB630;
```

```
T0PR=1;
T0TCR=1;
while(T0TC!=T0MR0);
T0TCR=2;
T0TC=0;
}
int main(void){
T0MCR=4;
IODIR1=0x00010000;
while(1){
IOSET1=1<<16;
on_delay();
IOCLR1=1<<16;
off_delay();
}
}
```

7.2 Generate a square wave using Timer0 in the interrupt mode.

```
# Write code here
#include<LPC214X.h>
unsigned int x = 0;
__irq void Timer0_ISR (void){
x ^= 1;
if(x)
IOSET1 = 1 << 20;
else
IOCLR1 = 1 << 20;
T0IR = 0x01;
VICVectAddr = 0x00000000;
}
int main(){
IODIR1 = 0x0FFFFFFF;
T0MCR = 0x00000003;
T0MR0 = 0x3456FF;
VICVectAddr4 = (unsigned)Timer0_ISR;
VICVectCntl4 = 0x00000024;
VICIntEnable = 0x00000010;
T0TCR = 1;
for(;;);
}
```

8. Exercises on Interfacing Keyboard to 8051

8.1 C Program to 4 X 4 matrix keypad using 8051

A keypad is a set of buttons arranged in a block or “pad” which usually bear digits, symbols and usually a complete set of alphabetical letters. If it mostly contains numbers then it can also be called a numeric keypad. Here we are using 4 X 4 matrix keypad.

In case of 4X4 matrix Keypad both the ends of switches are connected to the port pin i.e. four rows and four columns. So in all sixteen switches have been interfaced using just eight lines.Keypads arranged by matrix format, each row and column section pulled by high or low by selection J5, all row lines(P2.4 – P2.7) and column lines(P2.0 to P2.3) connected directly by the port pins.

Hint:

```
# Write program here
#include

//Define I/O Functions

#include

//Define 8051 Registers

#define DATA P0

//Define DATA to Port1 void lcd_init(void);

-
```

Try: 4×8 matrix keypad using 8051

Hint: Interface a 4×8 keypad with STM32 Blue Pill and program it in STM32CubeIDE using HAL libraries.

```
#In the first line, I'm assigning high to all columns. (c1=c2=c3=c4=1;)
Then I'm assigning the first row to zero and keeps the remaining row as high.
(r1=0;r2=1;r3=1;r4=1;)
----
----
----

#Then I'm checking the first column is zero or not. If it is zero then I should wait
until that button depressed. Then I can know the pressed key.
----
----
----

#If not I'm checking the next column. Like that, I'm checking all rows and columns.
If no keys pressed in row1, then I'm making row2 as zero. The remaining rows are
high. Then follow the above steps.
---
```

9. Exercises on interfacing of seven segment display, LDR, PIR sensors using Arduino

9.1 Interfacing Seven Segment Display with Arduino

The seven-segment displays are designed for displaying numeric values. You can find them anywhere from instruments to space shuttles. They are the most practical way to display numeric values.

Design Seven Segment Display for displaying numeric values using Arduino.

```
#include "SevSeg.h"
SevSeg sevseg;
void setup()
{
  byte numDigits = 1;

  byte digitPins[] = {};

  byte segmentPins[] = {9,8, 7, 6, 5, 4, 3, 2};
   byte displayType = COMMON_CATHODE;
   bool resistorsOnSegments = true;
   sevseg.begin(displayType, numDigits, digitPins, segmentPins, resistorsOnSegments);
   sevseg.setBrightness(90);
}
void loop()
{

   for(int i = 0; i <= 10; i++)
   {
     if (i == 10)
{
 i = 0;
}
     sevseg.setNumber(i);
     sevseg.refreshDisplay();
     delay(1000);
   }
}
```

9.2 Write the program for Automatic Night Light (LDR or Light Sensor)

Light Dependent Resistor (LDR) is the analog sensor that changes its resistance value or we can say the flow of current based on the light that falls on it. We can also use the Light Sensor Module instead of a single LDR.

Develop an Automatic night lights use a light-dependent resistor (LDR) to detect changes in ambient light levels.

```
# Write program here
const int ldrPin = A3;   // LDR analog pin

const int ledPin = 5;    // LED pin

int ldrThreshold = 500;  // Adjust this value based on ambient light

void setup() {

  pinMode(ledPin, OUTPUT);

  Serial.begin(9600);

}

void loop() {
```

```
  int ldrValue = analogRead(ldrPin);  // Read LDR value
  Serial.print("LDR Value: ");
  Serial.println(ldrValue);
  if (ldrValue < ldrThreshold) {
    digitalWrite(ledPin, HIGH);   // Turn on the LED when it's dark
    Serial.println("Dark. LED ON.");
  } else {
    digitalWrite(ledPin, LOW);    // Turn off the LED when it's bright
    Serial.println("Bright. LED OFF.");
  }
delay(1000);  // Delay before next reading
}
```

9.3 Detects the motion of objects and humans with the help of changes in infrared radiations.

A Passive Infrared Sensor or PIR is used to detect the motion of a specific object or human. It is widely used in applications where we want to take actions based on motion, and detection.

Implement prototype for detecting the motion of objects and humans with the help of changes in infrared radiations.

Hint: Use PIR sensor for detecting motion of objects.

```
# Write program here

const int pirPin = 2;    // PIR sensor's output pin

const int ledPin = 5;   // LED pin

int pirState = LOW;      // Initialize PIR sensor state

int lastPirState = LOW;  // Previous PIR sensor state

void setup() {

  pinMode(pirPin, INPUT);

  pinMode(ledPin, OUTPUT);

  Serial.begin(9600);

}
void loop() {

  pirState = digitalRead(pirPin);  // Read PIR sensor value


  if (pirState == HIGH && lastPirState == LOW) {
```

```
    // PIR sensor detects motion
    digitalWrite(ledPin, HIGH);   // Turn on the LED
    Serial.println("Motion detected!");
  } else if (pirState == LOW && lastPirState == HIGH) {
    // PIR sensor no longer detects motion
    digitalWrite(ledPin, LOW);    // Turn off the LED
    Serial.println("Motion stopped.");
  }
  lastPirState = pirState;  // Store current PIR state
}
```

10. Exercises on data acquisition from sensors using Arduino

10.1 Identification of obstacle using ultrasonic sensor

Ultrasonic is a very popular sensor recommended for all beginners to start their journey with sensors. It is a very easy-to-use sensor that can be used for distance measurement between objects and finding range.

Implement prototype for detecting the motion of objects and humans with the help of changes in infrared radiation.

```
#include<lpc214x.h>
const int trigPin = 3;   // Trigger pin of the ultrasonic sensor
const int echoPin = 2;  // Echo pin of the ultrasonic sensor
void setup() {
Serial.begin(9600);     // Initialize serial communication
pinMode(trigPin, OUTPUT);
pinMode(echoPin, INPUT);
}
void loop() {
digitalWrite(trigPin, LOW);
delayMicroseconds(2);
digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);
long duration = pulseIn(echoPin, HIGH);
float distance_cm = duration * 0.034 / 2;  // Calculate distance in centimeters
Serial.print("Distance: ");
```

```
Serial.print(distance_cm);
Serial.println(" cm");
delay(1000);  // Delay before next measurement
}
```

10.2 Identification of obstacle using IR ranger.

Infrared or IR proximity sensors emit infrared light and once this light hits an object, it is reflected back to the sensor. Depending on the strength of the reflected light, the sensor will know how far or close an object is. The stronger the reflected signal, the closer the object.

Implement prototype for detecting the object using IR ranger.

```
#include<lpc214x.h>

// Arduino IR Sensor Code
int IRSensor = 9; // connect ir sensor module to Arduino pin 9
int LED = 13; // conect LED to Arduino pin 13
void setup()
{
  Serial.begin(115200); // Init Serila at 115200 Baud

  Serial.println("Serial Working"); // Test to check if serial is working or not
  pinMode(IRSensor, INPUT); // IR Sensor pin INPUT
  pinMode(LED, OUTPUT); // LED Pin Output
}
void loop()
{
  int sensorStatus = digitalRead(IRSensor); // Set the GPIO as Input
  if (sensorStatus == 1) // Check if the pin high or not
  {
    // if the pin is high turn off the onboard Led
    digitalWrite(LED, LOW); // LED LOW
    Serial.println("Motion Ended!"); // print Motion Detected! on the serial monitor
window
  }
  else
  {
    //else turn on the onboard LED

    digitalWrite(LED, HIGH); // LED High
Serial.println("Motion Detected!"); // print Motion Ended! on the serial monitor
window
  }
}
```

10.3 Measures soil moisture by electrical conductivity changes

Soil moisture sensors measure or estimate the amount of water in the soil. These sensors can be stationary or portables such as handheld probes. Stationary sensors are placed at the predetermined locations and depths in the field, whereas portable soil moisture probes can measure soil moisture at several locations

Design a prototype to find dry or wetness of soil using soil moisture sensor. If the soil is dry pump the water using motor.

```
#include<lpc214x.h>

const int moisturePin = A3;  // Analog pin for soil moisture sensor
void setup() {
 Serial.begin(9600);     // Initialize serial communication

}
void loop() {
  int moistureValue = analogRead(moisturePin);  // Read soil moisture sensor value
  Serial.print("Moisture Level: ");

  Serial.println(moistureValue);
  delay(1000);  // Delay before next reading

}
```

Try: Detects flames by sensing infrared light emitted by flames 4

Hint: A flame detection sensor is a type of sensor that can detect and respond to the presence of a flame. Flame detectors can detect heat, smoke, and fire.

Flame detectors can detect flames in the 760–1100 nanometer wavelength range. They can detect small flames like a lighter flame at roughly 0.8m. The detection angle is roughly 60 degrees

```
# Write code here
// Flame Detection System
# Write code here

// Initialize serial communication
# Write code here

// Read flame sensor value
# Write code here

// Adjust threshold value based on flame detection
# Write code here
```

11. Exercises on weather monitoring system using Arduino.

11.1 Measurement of temperature using LM35

LM35 is a temperature sensor which can measure temperature in the range of -55°C to 150°C. It is a 3-terminal device that provides analog voltage proportional to the temperature. Higher the temperature, higher is the output voltage. The output analog voltage can be converted to digital form using ADC so that a microcontroller can process it.

Measure precise centigrade temperature by using LM35 and display the output temperature in Celsius form. It Increments the output by 1 on every 10-mV change in temperature.

```
# Write code here

const int lm35_pin = A1;  /* LM35 O/P pin */

void setup() {
  Serial.begin(9600);
}

void loop() {
  int temp_adc_val;
  float temp_val;
  temp_adc_val = analogRead(lm35_pin);  /* Read Temperature */
  temp_val = (temp_adc_val * 4.88);    /* Convert adc value to equivalent voltage */
  temp_val = (temp_val/10);      /* LM35 gives output of 10mv/°C */
  Serial.print("Temperature = ");
  Serial.print(temp_val);
  Serial.print(" Degree Celsius\n");
  delay(1000);
}
```

11.2 Measurement of humidity using DHT11

LM35 is a temperature sensor which can measure temperature in the range of -55°C to 150°C. It is a 3-terminal device that provides analog voltage proportional to the temperature. Higher the temperature, higher is the output voltage. The output analog voltage can be converted to digital form using ADC so that a microcontroller can process it.

Measure the temperature and humidity using DHT11 sensor to calculate electrical resistance between two electrodes.

```
# Write code here

// Digital Thermometer and Humidity Monitor (DHT11 or DHT22)
#include <DHT.h>
#define DHTPIN 2          // Digital pin connected to the DHT sensor
#define DHTTYPE DHT22     // DHT sensor type (DHT11 or DHT22)
DHT dht(DHTPIN, DHTTYPE); // Initialize DHT sensor
void setup() {
  Serial.begin(9600);     // Initialize serial communication
  dht.begin();            // Initialize DHT sensor
}
void loop() {
```

```
  // Read temperature and humidity from the sensor
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();
 // Print temperature and humidity values to Serial Monitor
  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.print(" °C\t");
  Serial.print("Humidity: ");
  Serial.print(humidity);
  Serial.println(" %");
  delay(2000);  // Delay before next reading
}
```

Try: Display the measured temperature and humidity on LCD

Hint: To measure the temperature and humidity, a thermistor and a capacitive humidity sensor are used, respectively. The resistance of a thermistor changes with a change in temperature - as the temperature increases, the resistance decreases. For the humidity sensor, the resistance between the two electrodes changes with a change in humidity

```
# Write code here
// Display data on 20x4 LCD
//Interface the DHT11 Temp & Humidity sensor and display humidity and  temperature
//in Celsius, Fahrenheit, and Kelvin on a 20x4 character LCD

# Write code here
--
double tempK = 0;
const int RS = 2, EN = 3, D4 = 4, D5 = 5, D6  = 6, D7 = 7;
# Write code here
-
LiquidCrystal lcd(RS,EN,D4,D5,D6,D7);
# Write code here
--
//set  20 columns and 4 rows of 16x2 LCD
# Write code here
```

12. Exercises on Arithmetic operations using ARM

12.1 Introduction to Keil MicrovisionV

- Install Keil MicrovisionV IDE
- LPC2148 Trainer kit
- Create a project, Edit an ASM file, Build, and Debug. Observe Disassembly window, Register and Memory contents in Step mode and in Run Mode.
- Execute a sample ARM Assembly Language Program to add two numbers in registers and store the sum in a register

12.2 Arithmetic Operations

- The arithmetic instructions include addition, subtraction, multiplication, division, comparison, negation, increment, and decrement.

Write a assembly level language program using arithmetic instructions to add first 5 natural numbers and store sum in register.

Preparation: ARM7, LPC2148 Trainer kit

```
find_addition of two numbers:
# Write code here
AREA PROG1,CODE,READONLY
ENTRY
MOV R0,#0x78
MOV R1,#0x21
ADD R3,R1,R0
STOP B STOP
END

find_addition of first 5 natural numbers:
# Write code here
AREA PROG1,CODE,READONLY
ENTRY
       MOV R0,#0x78
       MOV R1,#0x21
       ADD R3,R1,R0
       STOP B STOP
END

find_ add the first n even number:
# Write code here
AREA PROG6,CODE,READONLY
N RN 1
RESULT RN 2
EVEN_NUMBER RN 3
ENTRY
       MOV N,#5
       MOV RESULT,#0
       MOV EVEN_NUMBER,#2
# Write code here
--
--
--
```

```
LOOP ADD RESULT,RESULT,EVEN_NUMBER
# Write code here
--
--
--
--
STOP B STOP
END
```

find_ sum of cubes of the first n natural numbers:

```
# Write code here
AREA PROG9,CODE,READONLY
N RN 1
NPLUSONE RN 2
# Write code here
--
--
--
RESULT RN 4
ENTRY
       MOV R5,#0x40000000
       LDR N,=3
# Write code here
--
---
       MUL TEMP,N,NPLUSONE
       MOV TEMP,TEMP,LSR #1
       MUL RESULT,TEMP,TEMP
# Write code here
--
STOP   B STOP
END
```

12.3 Find the sum of cubes of the first n natural numbers

The sum of cubes of n natural numbers means finding the sum of a series of cubes of natural numbers. It can be obtained by using a simple formula S = [n2 (n + 1)2]/4, where S is the sum and n is the number of natural numbers taken.

Write an assembly level language program to find sum of cubes of the first n natural numbers.

Preparation: ARM7, LPC2148 Trainer kit

```
# Write code here
AREA PROG9,CODE,READONLY
N RN 1
NPLUSONE RN 2
TEMP RN 3
RESULT RN 4
ENTRY
MOV R5,#0x40000000
LDR N,=3
ADD NPLUSONE,N,#1
MUL TEMP,N,NPLUSONE
MOV TEMP,TEMP,LSR #1
MUL RESULT,TEMP,TEMP
```

```
STR RESULT,[R5]
STOP B STOP
END
```

12.4 Add the first n even numbers

The sum of the first n even numbers is n(n+1) This can be proven by using the sum of terms in an arithmetic progression formula.

Write an assembly level language program to add the first n even numbers. Store the result in a memory location .

Preparation: ARM7, LPC2148 Trainer kit

```
# Write code here
AREA PROG6,CODE,READONLY
N RN 1
RESULT RN 2
EVEN_NUMBER RN 3
ENTRY
       MOV N,#5
       MOV RESULT,#0
       MOV EVEN_NUMBER,#2
       MOV R4,#0x40000000
LOOP ADD RESULT,RESULT,EVEN_NUMBER
       ADD EVEN_NUMBER,EVEN_NUMBER,#2
SUBS N,N,#1
BNE LOOP
STR RESULT,[R4]
STOP B STOP
END
```

Try: Write an ALP to compute sum of squares of 5 numbers starting from 1.

Hint: Sum of squares refers to the sum of the squares of numbers. It is basically the addition of squared numbers. The squared terms could be 2 terms, 3 terms, or 'n' number of terms, first n even terms or odd terms, set of natural numbers or consecutive numbers, etc.

```
# Write code here
# (x1)2+(x2)2+….+(xn)2→Sum of squares of n numbers
# Write code here
--
AREA PROG5,CODE,READONLY
ENTRY
# Write code here
-
-
LOOP BL SQU
CMP R2,#6
# Write code here
-
-
SQU MUL R4,R2,R2
```

```
# Write code here
```

13. Exercises on interfacing peripheral devices to ARM7 LPC2148 Trainer kit

13.1 Toggle LED's with some time delay

LED blinking is the process of continuously turning an LED (Light Emitting Diode) on and off in a repetitive pattern. The basic concept involves toggling the state of the LED between ON and OFF at a specific rate, creating a blinking effect.

Write a program to make a LED glow at different brightness levels (low to high) with brightness levels varying over duration of 2s.

```
# Write code here
# include <lpc214x.h>
voidpwm_init(void)
{
PINSEL0|=0x00000002;
PWMPR= 0x2;
PWMPCR=0x00000200;
PWMMR0=0xC37F;
PWMMCR=0x00000002;
PWMTCR=0x00000009;
}
int main()
{
inti;
pwm_init();
while(1)
{ for(i=0;i<10;i++)
{PWMMR1=0xFFF+(0xFF5*i);
PWMLER=0x02;
}}}
```

13.2 Interface keyboard and LCD

Interfacing an LCD with an ARM7 microcontroller typically involves using a combination of GPIO (General Purpose Input/Output) pins to control the data and command lines of the LCD. Below is a simple example of an embedded C program to interface an ARM7 microcontroller with a 16x2 character LCD using a 4-bit mode connection.

To develop and verify the interfacing of keyboard and LCD with ARM development kit using embedded C program.

For_ Display a text in 8 bit LCD using LPC2148 -ARM7 Advanced Development Board:

```
# Write code here
#include
#include
```

```
#include "LCD8.H"
void main()
{
PINSEL0 |= 0x05;
PINSEL1 = 0;
LCD_Config (&IOPIN0, 16);
while(1)
       //Loop From Here....
{
Delay();     # delay

Delay();
lcd_cmd (&IOPIN0,16, 0x01);
Delay();
lcd_data (&IOPIN0,16,'1');
}
}
void delay(unsigned int n)
{
unsigned int i,j;
for(i=0;i<n;i++)
for(j=0;j<12000;j++);
}
```

Try: Write an embedded C program for interfacing LED and PWM and to verify the output in the ARM kit

To verify the flashing of LEDS in ARM DEVELOPMENT KIT microcontroller board using embedded C program.

Hint: A light that goes on and off in a specific pattern. The pattern can be fast or slow and the lights can be a solid color or a variety of different colors.

```
For_flashing of LEDS
# Write code here
#include <lpc214x.h>
unsigned int delay;
int main(void)
{
 IO0DIR = (1<<10); // Configure P0.10 as Output
 while(1)
 {
 IO0CLR = (1<<10); // CLEAR(0)P0.10toturn LED ON
 for(delay=0; delay<500000; delay++); // delay
 IO0SET = (1<<10); // SET (1) P0.10 to turn
LEDs OFF
 for(delay=0; delay<500000; delay++); // delay
 }

For_PWM
```

```
# Write code here
--
--
--
# Write code here
--
--
if (val != oldval)
 {
 PWMMR2 = val;
 PWMLER = 0xF;
 oldval = val;
# Write code here
--
--
```

14. Case study: ARM7/ARM9 prototype design.

14.1 Build a smoke detector using an MQ-2 gas sensor.

Materials:

- MQ-2 gas sensor
- Arduino board
- Breadboard
- Jumper wires
- Buzzer
- LED
- Resistor (10k ohms)

```
# Write code here
int sensorPin = A0;
int ledPin = 3;
int buzzerPin = 2;
int sensorValue = 0;
-
# Write code here
-
void loop() {
  sensorValue = analogRead(sensorPin);

# Write code here
-
# Write code here
  }
}
```

14.2 Build an IoT-based water quality monitoring system without any TDS/pH meter

DS stands for Total Dissolved Solids. As the name suggests, it gives us the number of solids dissolved in a certain amount of water, in ppm (parts per million). TDS is calculated based on electrical conductivity [S/m]. The higher the electrical conductivity, the higher the TDS value. Here is a list of the TDS values of different **types of water:**

Pure water: 80-150

Tap water: 250-350

Groundwater: 500-1000

Seawater: around 30000

```
# Write code here
//include libraries
#include <SoftwareSerial.h>
#include <LiquidCrystal.h>

//for  bluetooth - create an object called BTserial, with RX pin at 3 and TX pin at 2
# Write code here

//decraration of all our variables
# Write code here

float  reads;
int pin = A0;
# Write code here

//resistance between the 2 wires
# Write code here

//resistivity
# Write code here

//distance between the wires in m
# Write code here

//area of cross  section of wire in m^2
# Write code here

float C = 0;//conductivity in S/m
float Cm = 0;//conductivity  in mS/cm
# Write code here

int rPin = 9;
int bPin = 5;
int gPin = 6;
int rVal = 255;
int  bVal = 255;
```

```
int gVal = 255;
# Write code here
```

V. TEXT BOOKS:

1. Ray A.K, Bhurchandi K.M, "Advanced Microprocessor and Peripherals", TMH, 2nd Edition, 2012.
2. Muhammad Ali Mazidi, J.G. Mazidi, R.D McKinlay, "The 8051 Microcontroller and Embedded Systems using Assembly and C", Pearson education, 2nd Edition, 2009.
3. Douglas V. Hall, "Microprocessors and Interfacing Programming and Hardware", TMGH, 2nd Edition, 1994.

VI. REFERENCE BOOKS:

1. Kenneth J. Ayala, "The 8051 Microcontroller", Thomson Learning, 3rd Edition, 2005.
2. Manish K. Patel, "The 8051 Microcontroller Based Embedded Systems", McGraw Hill, 1st Edition, 2014.
3. Ajay V Deshmukh, "Microcontrollers", TATA McGraw Hill Publications, 2nd Edition, 2012.