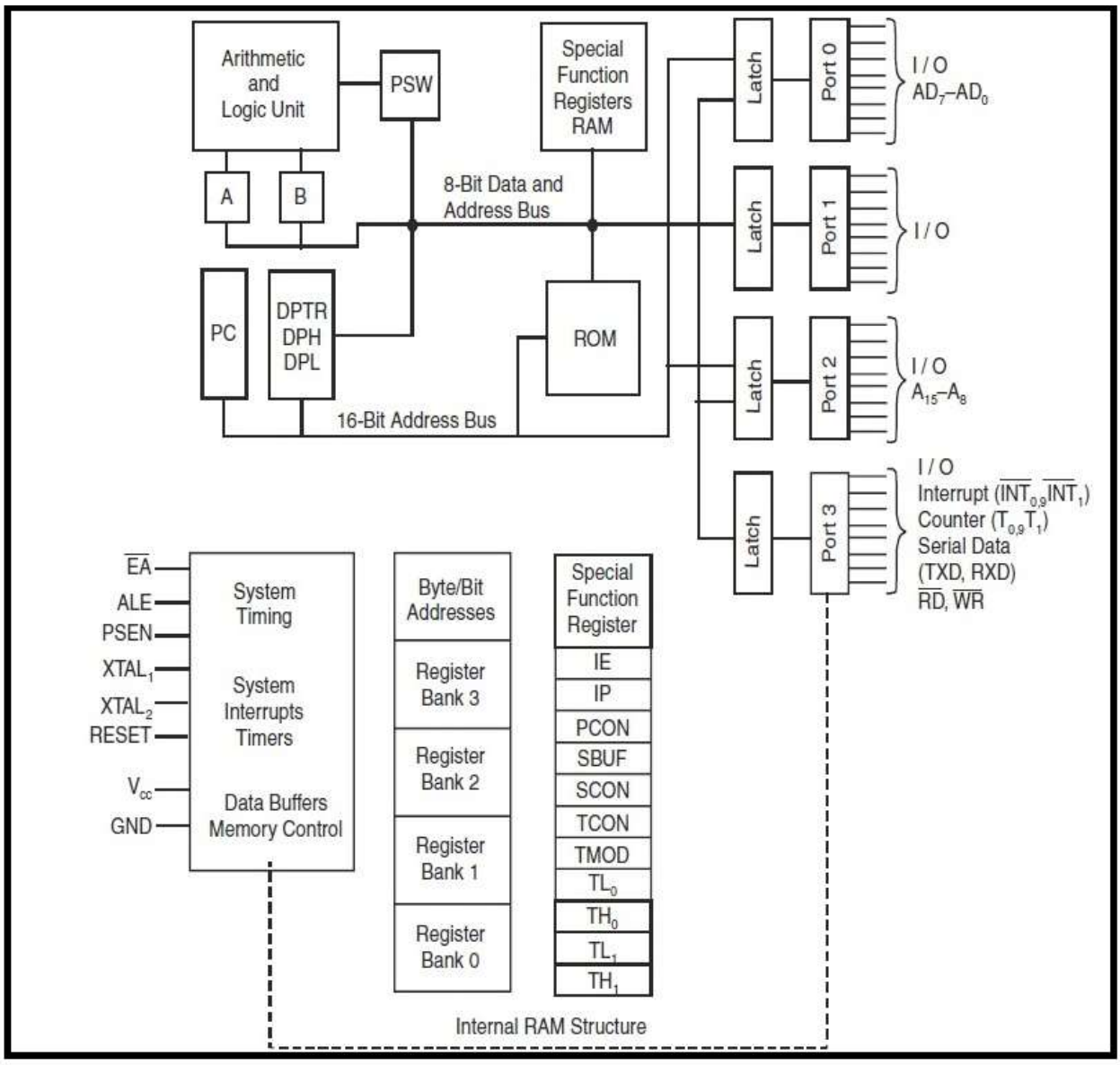
Date: 30/01/23 **Exp. No.:** 01	**STUDY OF 8051 MICROCONTROLLER AND INSTRUCTION SET**

AIM:

To Study the architecture and the instruction set of the 8051 microcontroller.

THEORY:

Architecture of 8051 Microcontroller:



Instruction Sets:

The instructions of the 8051 microcontrollers are divided into the five functional groups:

- Arithmetic operations
- Logical operations
- Data transfer operations
- Boolean variable operations
- Program branching operations

1. Arithmetic Operations:

Mnemonic	Description
ADD A, Rn	A = A + [Rn]
ADD A, direct	A = A + [direct memory]
ADD A, @Ri	A = A + [memory pointed to by Ri]
ADD A, #data	A = A + immediate data
ADDC A, Rn	A = A + [Rn] + CY
ADDC A, direct	A = A + [direct memory] + Cy
SUBB A, Rn	A = A - [Rn] - CY
SUBB A, #data	A = A - immediate data - CY
INC A	A = A + 1
INC Rn	[Rn] = [Rn] + 1
DEC A	A = A - 1
DEC Rn	[Rn] = [Rn] -1
DEC @Ri	[@Ri] = [@Ri] - 1
MUL AB	Multiply A and B
DIV AB	Divide A by B
DA A	Decimal adjust A

2. Logical Operations:

Mnemonic	Description
ANL A, Rn	A = A & [Rn]
ANL A, direct	A = A & [direct memory]
ANL direct, A	[direct] = [direct] & A
ANL direct, #data	[direct] = [direct] & immediate data
ORL A, Rn	A = A OR [Rn]
ORL A, direct	A = A OR [direct]
ORL A, @Ri	A = A OR [@Ri]
ORL direct, #data	[direct] = [direct] OR immediate data
XRL A, Rn	A = A XOR [Rn]

XRL A, @Ri	A = A XOR [@Ri]
XRL direct, #data	[direct] = [direct] XOR immediate data
XRL direct, A	[direct] = [direct] XOR A
CLR A	Clear A
CPL A	Complement A
RL A	Rotate A left
RLC A	Rotate A left (through C)
RR A	Rotate A right
SWAP A	Swap nibbles

3. Data Transfer Operations:

Mnemonic	**Description**
MOV @Ri, direct	[@Ri] = [direct]
MOV @Ri, #data	[@Ri] = immediate data
MOV DPTR, #data16	[DPTR] = immediate data
MOVC A, @A+DPTR	A = Code byte from [@A+DPTR]
MOVX A, @Ri	A = Data byte from external RAM[@Ri]
MOVX @Ri, A	External [@Ri] = A
PUSH direct	Push into stack
POP direct	Pop from stack
XCH A, Rn	A = [Rn], [Rn] = A
XCH A, direct	A = [direct], [direct] = A
XCHD A, @Ri	Exchange lower order digits

4. Boolean Variable Operations:

Mnemonic	**Description**
CLR C	Clear C
CLR bit	Clear direct bit
SETB C	Set C
CPL C	Complement C

CPL bit	Complement direct bit
ANL C, bit	AND bit with C
ORL C, bit	OR bit with C
MOV C, bit	MOV bit to C
JC rel	Jump if C set
JNC rel	Jump if C not set
JB bit, rel	Jump if specified bit set
JBC bit, rel	If specified bit set then clear and jump

5. Program Branching Operations:

Mnemonic	**Description**
ACALL addr11	Absolute subroutine call
LCALL addr16	Long subroutine call
RET	Return from subroutine
AJMP addr11	Absolute jump
SJMP rel	Short jump
JMP @A+DPTR	Jump indirect
JZ rel	Jump if A=0
JNZ rel	Jump if A not = 0
CJNE A, direct, rel CJNE A, #data, rel CJNE @Ri, #data, rel	Compare and Jump if not equal
DJNZ Rn, rel DJNZ direct, rel	Decrement and jump if not zero
NOP	No operation

RESULT:

Thus, the architecture and the instruction set of the 8051 microcontrollers has been studied successfully.

Date: 06/02/23 **Exp. No.:** 02	**8-BIT ARITHMETIC OPERATIONS**

AIM:

To perform 8-bit arithmetic operations using 8051 microcontrollers.

1. Addition:

ALGORITHM:

1) Move first value to accumulator.
2) Move the second value into register.
3) Add the second value to accumulator value.
4) Accumulator stores the sum of the two values.

CODE:

Address	OP Code	Mnemonics	Description
8500	74 25	MOV A, #025	Move immediate value 25 into accumulator
8502	78 27	MOV R0, #027	Move immediate value 27 into register R0
8504	28	ADD A, R0	Add accumulator value and R0 value
8505	12 00 BB	LCALL 00BB	End

OUTPUT:

A: 4C

2. Subtraction:

ALGORITHM:

1) Move first value to accumulator.
2) Move the second value into register.
3) Subtract the second value to accumulator value.
4) Accumulator stores the difference between the two values.

CODE:

Address	OP Code	Mnemonics	Description
8600	74 27	MOV A, #27	Move immediate value 27 into accumulator
8602	78 25	MOV R0, #25	Move immediate value 25 into register R0
8604	98	SUBB A, R0	Subtract value of R0 from accumulator
8605	12 00 BB	LCALL 00BB	End

OUTPUT:

A: 02

3. Multiplication using Repeated Addition:

ALGORITHM:

1) Move 00 into accumulator.
2) Move the second value into register R0.
3) Move value 00 into register R2.
4) Add the first value to accumulator value.
5) If carry generated, add value 01 to register R2.
6) Decrement value in R0 and jump to step 4 until value in R0 is not zero.
7) Move accumulator value into register R1.

CODE:

Address	OP Code	Mnemonics	Description
8700	74 7F	MOV A,#00	Move immediate value 0 into Acc
8702	78 05	MOV R0,#05	Move immediate value 05 into R0
8704	7A 00	MOV R2,#00	Move immediate value 0 to R2
8706	24 7F	ADD A,#7F	Add immediate value 7F to A value
8708	50	JNC 870E	Jump to address 870E if C is not set
870A	C9	XCH A,R1	Swap the values of A and R1
870B	74 01	ADD A,#01	Add immediate value 01 to A value
870D	C9	XCH A,R1	Swap the values of A and R1
870E	D8 8514	DJNZ R0,8706	Decrement R0 value and while not zero,
8710	F9	MOV R1,A	Move accumulator value to R1
8711	12 00 BB	LCALL 00BB	End

OUTPUT:

R2: 02 R1:7B

4. Multiplication without Registers:

ALGORITHM:

1) Move first value to accumulator
2) Move the second value into B register
3) Multiply the contents of accumulator and B register
4) The product of the two values is present in accumulator

CODE:

Address	OP Code	Mnemonics	Description
8800	74 02	MOV A, #7F	Move immediate value 2 into accumulator
8802	75 0F 03	MOV F0, #05	Move immediate value 3 into B register
8805	A4	MUL AB	Multiply contents of A and B
8806	12 00 BB	LCALL 00BB	End

OUTPUT:

B:02 A:7B

5. Multiplication with Registers:

ALGORITHM:

1) Move the multiplicand into register R1.
2) Move the multiplier into register R2.
3) Move the contents of R1 into the accumulator.
4) Move the contents of R2 into B register.
5) Multiply the contents of accumulator and B register.
6) Move the product, which is in accumulator, into register R3.
7) Move the carry present in B register into register R4.

CODE:

Address	OP Code	Mnemonics	Description
8900	79 10	MOV R1, #10	Move immediate value 10 into R1
8902	7A 05	MOV R2, #05	Move immediate value 5 into R2
8904	E9	MOV A, R1	Move contents of R1 into A
8905	8A F0	MOV F0, R2	Move contents of R2 into B
8907	A4	MUL AB	Multiply contents of A and B
8908	FB	MOV R3, A	Move the contents of A into R3
8909	AC F0	MOV R4, F0	Move the contents of B into R4
890B	12 00 BB	LCALL 00BB	End

OUTPUT:

R4:00 R3:50

6. Multiplication with Memory:

ALGORITHM:

1) Move the multiplicand into accumulator
2) Move the multiplier into B register
3) Multiply contents of accumulator and B register
4) Move the address of 1st instruction into DPTR
5) Move value in accumulator into address pointed to by DPTR
6) Increment the value in DPTR by one
7) Move contents of B to accumulator
8) Move contents of A to address pointed to by DPTR

CODE:

Address	OP Code	Mnemonics	Description
9000	74 03	MOV A, #03	Move immediate value 3 into accumulator
9002	75 F0 04	MOV F0, #04	Move immediate value 4 into B register
9005	A4	MUL AB	Multiply contents of A and B
9006	90 90 00	MOV DPTR, #8000	Move immediate value 9000 into DPTR
9009	F0	MOVX @DPTR, A	Move contents of A into address pointed to
900A	A3	INC DPTR	Increment DPTR
900B	E5 0B	MOV A, F0	Move contents of B to A
900D	F0	MOVX @DPTR, A	Move contents of A into address in DPTR
900E	12 00 B	LCALL 00BB	End

OUTPUT:

8000:0C 8001:00

7. Division with Registers:

ALGORITHM:

1) Move the dividend into register R1
2) Move the divisor into register R2
3) Move the contents of R1 into the accumulator
4) Move the contents of R2 into B register
5) Divide the contents of accumulator by that of B register
6) Move the quotient stored in accumulator into register R3
7) Move the remainder present in B register into register R4

CODE:

Address	OP Code	Mnemonics	Description
8500	79 04	MOV R1, #04	Move immediate value 4 into R1
8502	7A 02	MOV R2, #02	Move immediate value 2 into R2
8504	E9	MOV A, R1	Move contents of R1 into A
8505	8A F0	MOV F0, R2	Move contents of R2 into B
8507	84	DIV AB	Divide contents of A by that of B
8508	FB	MOV R3, A	Move the contents of A into R3
8509	AC F0	MOV R4, F0	Move the contents of B into R4
850B	12 00 BB	LCALL 00BB	End

OUTPUT:

R3:02 R4:00

RESULT:

Thus, 8-bit arithmetic operations have been executed using 8051 Microcontroller successfully.

Date: 13/02/23 **Exp. No.:** 03	**16-BIT ARITHMETIC OPERATIONS**

AIM:

To perform 16-bit arithmetic operations using 8051 microcontrollers.

1. Addition with DPTR:

ALGORITHM:

1.Split the 16-bit numbers to be added into 8-bit numbers.
2.Store the 8-bit numbers in individual registers.
3.Add the corresponding numbers and store the values in each register.
4.The registers put together gives the result.

CODE:

Address	OP Code	Mnemonics	Description
8500	78 40	MOV R0, #40	Move immediate value 40 to R0
8502	E6	MOV A, @R0	Move contents of R0 to A
8503	08	INC R0	Increment R0
8504	26	ADD A, @R0	Add contents of R0 to A
8505	90 86 00	MOV DPTR, #8600	Move immediate value 8600 to DPTR
8508	F0	MOVX @DPTR, A	Move A value to DPTR
8509	A3	INC DPTR	Increment DPTR
850A	08	INC R0	Increment R0 value
850B	E6	MOV A, @R0	Move contents of R0 to A
850C	08	INC R0	Increment R0
850D	36	ADDC A, @R0	Add contents of R0 to A
850E	F0	MOVX @DPTR, A	Move contents of A to DPTR
850F	12 00 BB	LCALL 00BB	End

OUTPUT:

40: 07 42: 08
41: 0F 43: 0F
8600: 16 8601: 17

2. Subtraction with Registers:

ALGORITHM:

1. Move value 0 into register R1
2. Move value 0 into register R2
3. Move value into register to R3
4. Move contents of R1 into accumulator
5. Subtract the value of R2 from that of Accumulator. The difference is stored in Accumulator.
6. Move value of Accumulator into register R6
7. Move contents of register R3 into Accumulator
8. Subtract value of R4 from that of Accumulator. The difference is stored in Accumulator.
9. Move value in Accumulator into register R5

CODE:

Address	OP Code	Mnemonics	Description
8700	79 00	MOV R1, #00	Move immediate value 0 to R1
8702	7A 00	MOV R2, #00	Move immediate value 0 to R2
8704	7B 50	MOV R3, #50	Move immediate value 50 to R3
8706	7C 30	MOV R4, #30	Move immediate value 30 to R4
8708	E9	MOV A, R1	Move contents of R1 to A
8709	9A	SUBB A, R2	Subtract R2 from A and store in A
870A	FE	MOV R6, A	Move contents of A to R6
870B	EB	MOV A, R3	Move contents of R3 to A
870C	9C	SUBB A, R4	Subtract A from R4 and store in A
870D	FD	MOV R5, A	Move contents of A to R5
870E	12 00 BB	LCALL 00BB	End

OUTPUT:

R3: 50 R1: 00
R4: 30 R2: 00
R5: 20 R6: 00

3. Multiplication:

ALGORITHM:

1.Split the 16-bit data into 8-bit numbers
2.Store each 8-bit number in a register
3.Move first 8-bits of multiplier into Accumulator
4.Move next 8-bits of multiplier into B register
5.Multiply contents of A and B
6.Move the accumulator's contents into register R6
7.Move the contents of B register into register R6
8.Move the contents of R4 into Accumulator
9.Move content of register R1 into B register
10.Multiply values in A and B
11.Move the value in B register into register R7
12.Add value in register R6 to Accumulator
13.Save value in Accumulator in register R6
14.Move value in register R2 into Accumulator
15.Move value in register R3 into B register
16.Multiply values in A and B
17.Add values in Accumulator and register R6
18.Move Accumulator value into register R6
19.Move B register value into Accumulator
20.Add values present in register R7 and Accumulator
21.Move Accumulator value into register R7
22.Move value of R1 into Accumulator
23.Move value of R2 into B register
24.Multiply values in A and B
25.Add values in register R7 to that of Accumulator
26.Move value in Accumulator into register R7
27.Move value in B register into Accumulator
28.Add Accumulator value and 00 with previous carry
29.Move value in Accumulator into register R0

CODE:

Address	OP Code	Mnemonics	Description
8600	79 3F	MOV R1, #3F	Move immediate value 3F into R1
8602	7A 23	MOV R2, #23	Move immediate value 23 into R2
8604	7B 11	MOV R3, #11	Move immediate value 11 into R3
8606	7C F2	MOV R4, #F2	Move immediate value F2 into R4
8608	EB	MOV A, R3	Move R3 contents to A
8609	8C F0	MOV F0, R4	Move R4 contents to B
860B	A4	MUL AB	Multiply A and B
860C	FD	MOV R5, A	Move contents to R5
860D	AE F0	MOV R6, F0	Move B contents to R6
860F	EC	MOV A, R4	Move R4 contents to A
8610	89 F0	MOV F0, R1	Move R1 contents to F0
8612	A4	MUL AB	Multiply A and B
8613	AF F0	MOV R7, F0	Move B contents to R7
8615	3E	ADDC A, R6	Add R6 contents to A and store it
8616	FE	MOV R6, A	Move A contents to R6
8617	EA	MOV A, R2	Move R2 contents to A
8618	8B F0	MOV F0, R3	Move R3 contents to B
861A	A4	MUL AB	Multiply A and B
861B	3E	ADDC A, R6	Add contents of A and R6
861C	FE	MOV R6, A	Move A contents to R6
861D	E5 F0	MOV A, F0	Move B contents to A
861F	3F	ADDC A, R7	Add R7 and A contents in A
8620	FF	MOV R7, A	Move A contents to R7
8621	E9	MOV A, R1	Move R1 contents to A
8622	8A F0	MOV F0, R2	Move R2 contents to B
8624	A4	MUL AB	Multiply A and B
8625	3F	ADDC A,R7	Add A and R7 with Carry
8626	FF	MOV R7, A	Move A contents to R7
8627	E5 F0	MOV A, F0	Move B contents to A
8629	34 00	ADDC A, #00	Add A and 00 with previous carry
862B	F8	MOV R0, A	Move A contents to R0
862C	12 00 BB	LCALL 00BB	End

OUTPUT:

R0: 08 R5: 12 R6: F1 R7: DA

RESULT:

Thus, 16-bit arithmetic operations have been executed using 8051 Micro Controller successfully.

Date: 13/02/23 **Exp. No.:** 04	**MULTIBYTE ARITHMETIC OPERATIONS**

AIM:

To perform multibyte arithmetic operations using 8051 microcontrollers.

1. Addition:

ALGORITHM:

1. Initialize R0 with 1st bit of 1st operand
2. Initialize R1 with 2nd operand
3. Initialize R3 with counter value
4. Move value in R0 to accumulator
5. Add value in R1 to accumulator
6. Store sum in R1
7. Increment values of R0 and R1
8. Decrement R3 and go to step 4 if value in R3 is more than 0

CODE:

Address	OP Code	Mnemonics	Description
8500	78 40	MOV R0, #40	Move 40 to R0
8502	79 50	MOV R1, #50	Move 50 to R1
8504	7B 04	MOV R3, #04	Move immediate value 4 to R3
8506	E6	MOV A, @R0	Move R0's address value to A
8507	97	ADDC A, @R1	Add R1 address value to A
8508	F7	MOV @R1, A	Move A to R1
8509	08	INC R0	Increment R0 value
850A	09	INC R1	Increment R1 value
850B	DB F9	DJNZ R3,8506	Decrement R3 and go to 8506 if R3>0
850D	12 00 BB	LCALL 00BB	End

INPUT:

40: 03 41: 04 42: 05 43: 06
50: 04 51: 04 52: 05 53: 06

OUTPUT:

50: 06 51:08 52: 0A 53: 0C

2. Subtraction:

ALGORITHM:

1. Initialize R0 with 1st bit of 1st operand address
2. Initialize R1 with 1st bit of 2nd operand address
3. Initialize R2 as counter
4. Move value in R0 into accumulator
5. Subtract value in accumulator by value in R1
6. Move value in accumulator into register R1
7. Increment values of R0 and R1
8. Decrement value of R2 and go to step 4 if value in R2 is more than 0

CODE:

Address	OP Code	Mnemonics	Description
8700	78 40	MOV R0, #40	Move 40 to R0
8702	79 50	MOV R1, #50	Move 50 to R1
8704	7A 04	MOV R2, #04	Move immediate value 4 to R2
8706	C9	CLR C	Clear carry
8707	E6	MOV A, @R0	Move value at R0 to accumulator
8708	97	SUBB A, @R1	Subtract A value with value in R1
8709	F7	MOV @R1, A	Move A value to R1
870A	08	INC R0	Increment R0
870B	09	INC R1	Increment R1
870C	0A F9	DJNZ R2,8707	Go to 8707 if R2 is positive after decrement
870E	12 00 BB	LCALL 00BB	End

INPUT:

40: 09 41: 08 42: 07 43: 06
50: 03 51: 02 52: 04 53: 05

OUTPUT:

50: 06 51: 06 52: 03 53: 01

3. Multiplication:

ALGORITHM:

1. Store the first 16-bit operand at address R2
2. Store the multiplier at other registers
3. Initialize value 0 to R4 register
4. Clear the carry
5. Move the value in R0 to accumulator
6. Move the value in R5 to register B
7. Multiply values in A and B
8. Add R4 value to the accumulator and carry
9. Move accumulator value to register R1
10. Move register B value to register R4
11. Increment value of R0 and R1
12. Decrement value of R2 and go to step 5 if greater than 0
13. Move B register value into accumulator
14. Add carry value to accumulator
15. Move accumulator value into R1

CODE:

Address	OP Code	Mnemonics	Description
8850	7A 02	MOV R2, #02	Move immediate value 2 to R2
8852	78 40	MOV R0, #40	Move immediate value 40 to R0
8854	A0 50	MOV R5, #50	Move immediate value 50 to R5
8856	79 60	MOV R1, #60	Move immediate value 60 to R1
8858	C3	CLRC	Clear carry
8859	7C 00	MOV R4, #00	Initialize R4 value as 0

885A	E6	MOV A, @R0	Move indirect content of R0 to A
885C	8D F0	MOV F0, R5	Move R5 content to register B
885E	A4	MUL AB	Multiply A and B
885F	3C	ADDC A, R4	Add R4 value to accumulator and carry
8860	F7	MOV @R1, A	Move A value to indirect content of R1
8861	AC F0	MOV R4, F0	Move B register value to R4
8863	08	INC R0	Increment R0 value
8864	09	INC R1	Increment R1 value
8865	DA F4	DJNZ R2,885A	Jump to 885B if R2>0 after decrement
8867	E5 F0	MOV A, F0	Move F0 value to accumulator
8869	34 00	ADDC A, #00	Add A with carry
886B	F7	MOV @R1, A	Move A to indirect content of R1
886C	12 00 BB	LCALL 00BB	End

INPUT:

40: 53 41: 49 42: A2

OUTPUT:

60: F0 61: E9 62: B6 63: 32

RESULT:

Thus, multibyte arithmetic operations have been executed using 8051 Micro Controller successfully.

Date: 20/02/23 **Exp. No.:** 05	**SORTING PROGRAM**

AIM:

To sort the given elements in the ascending and descending order.

1. Descending Order:

ALGORITHM:

1. Store the array of elements in the internal memory
2. Sort the given elements in descending order and store it in the same memory
3. Display the result

CODE:

Address	Mnemonics
8500	MOV R4, #05
8502	MOV R3, #05
8504	MOV R0, #40
8506	CLR C
8507	MOV A, @R0
8508	MOV R1, A
8509	INC R0
850A	MOV A, @R0
850B	SUBB A, R1
850C	JC 8516
850E	MOV A, @R0
850F	DEC R0
8510	MOV @R0, A
8512	MOV A, R1
8513	INC R0
8514	MOV @R0, A
8516	DJNZ R3,8507
8518	DJNZ R4, 8502
851A	LCALL 00BB

INPUT:

40: 02 41: 03 42: 08 43: 09 44: 01

OUTPUT:

40: 09 41: 08 42: 03 43: 02 44: 01

2. Ascending Order:

ALGORITHM:

1. Store the array of elements in the internal memory
2. Sort the given elements in ascending order and store it in the same memory
3. Display the result

CODE:

Address	Mnemonics
8500	MOV R4, #05
8502	MOV R3, #05
8504	MOV R0, #40
8506	CLR C
8507	MOV A, @R0
8508	MOV R1, A
8509	INC R0
850A	MOV A, @R0
850B	SUBB A, R1
850C	JNC 8516
850E	MOV A, @R0
850F	DEC R0
8510	MOV @R0, A
8512	MOV A, R1
8513	INC R0
8514	MOV @R0, A
8516	DJNZ R3,8507
8518	DJNZ R4, 8502
851A	LCALL 00BB

INPUT:

40: 02 41: 03 42: 08 43: 09 44: 01

OUTPUT:

40: 01 41: 02 42: 03 43: 08 44: 09

RESULT:

Thus, the sorting programs were implemented and output was verified.

Date: 20/02/23 **Exp. No.:** 06	**CODE CONVERTERS**

AIM:

To write assembly language programs for the following code conversions:

i) BCD to HEX ii) HEX to BCD iii) Binary to Gray Code
iv) Gray Code to Binary v) ASCII to HEX

1. BCD to HEX:

ALGORITHM:

1. Separate the first and last 4 bits by applying logical AND with 0F and F0
2. Swap the first and last 4 bits of part AND-ed with F0 to make it single digit
3. Multiply ten's place digit by 10 and add unit's place

CODE:

Address	Mnemonics
8800	MOV R0, #99
8802	MOV A, R0
8803	ANL A, #0F
8805	MOV R1, A
8806	MOV A, R0
8807	ANL A, #F0
8809	SWAP A
880A	MOV F0, #0A
880D	MUL AB
880E	ADD A, R1
880F	MOV R1, A
8810	LCALL 00BB

OUTPUT:

A: 63

2. HEX to BCD:

ALGORITHM:

1. Divide the numerator by 64. Store the quotient in R1.
2. Divide remainder by 10
3. Store quotient in R2 and remainder in R3

CODE:

Address	Mnemonics
8500	MOV A, #FF
8502	MOV F0, #64
8505	DIV AB
8506	MOV R1, A
8507	MOV A, F0
8509	MOV F0, #0A
850C	DIV AB
850D	MOV R2, A
850E	MOV R3, F0
8510	LCALL 00BB

OUTPUT:

R1: 02 R2:05 R3: 05

3. Binary to Gray Code:

ALGORITHM:

1. Store the binary value in a register and transfer to accumulator
2. Right Rotate accumulator and XOR it with value in the register

CODE:

Address	Mnemonics
8500	CLR C
8501	MOV A, #0A
8503	MOV R0, A
8504	RRC A
8505	XRL A, R0
8506	LCALL 00BB

OUTPUT:

A: 0F

4. Gray Code to Binary:

ALGORITHM:

1. Store the gray code value in a register and transfer to accumulator
2. Initialize a counter at value 7
3. Store accumulator value in B register. Retrieve MSB by AND-ing contents of A with 80H
4. Right-rotate accumulator and AND with 7F to retrieve next 7 bits. XOR bits with B register
5. Repeats steps till counter is positive

CODE:

Address	Mnemonics
8800	MOV R0, #0F
8802	MOV A, R0
8803	MOV R3, #07
8805	MOV F0, A
8807	RRC A
8808	XRL A, F0
880A	CLR C
880B	DJNZ R3,8807
880D	LCALL 00BB

OUTPUT:

A: 0A

5. ASCII to HEX:

ALGORITHM:

1. Store the data in a register and transfer to accumulator
2. Subtract 0A to check if it is in 0-9. If it is, add 30 to convert to ASCII. If it is in A-F, add 37

CODE:

Address	Mnemonics
8750	MOV R0,#0A
8752	MOV A,R0
8753	CRL C
8754	SUBB A,#0A
8756	JC 875C
8758	ADD A,#37
875A	SJMP 875F
875C	MOV A,R2
875D	ADD A,#30H
875F	MOV R1,A
8760	LCALL 00BB

OUTPUT:

R1: 41

RESULT:

Thus, the code conversions were performed using assembly language programs.

Date: 20/02/23 **Exp. No.:** 07	**ADDITION OF N-NUMBERS**

AIM:

To write assembly language programs to add n given numbers.

ALGORITHM:

1. Store the n values in address 50
2. Store the array value from 41 to n addresses
3. Initialize R3 to 0, move value from 41 to A, and store it in R2
4. Add A and R2 and store result in R3
5. Repeat the above steps until counter value not equal to 0

CODE:

Address	Mnemonics
8600	MOV A, #00
8602	MOV R0, #40
8604	MOV R1, #50
8606	MOV R2, @R1
8607	MOV R2, A
8608	MOV A, #00
860A	ADD A, @R0
860B	INC R0
860C	DJNZ R2,860A
860E	LCALL 00BB

INPUT:

40: 05 41: 08 42: 09 43: 0A 44: 0D

OUTPUT:

A: 20

RESULT:

Thus, the assembly language programs to add n user-given numbers has been written and executed successfully.

Date: 05/05/23 **Exp. No.:** 08	**TIMER DELAY USING ALP IN KEIL**

AIM:

To write an Assembly Language program to generate a time delay

1. Timer Delay program with no interrupts:

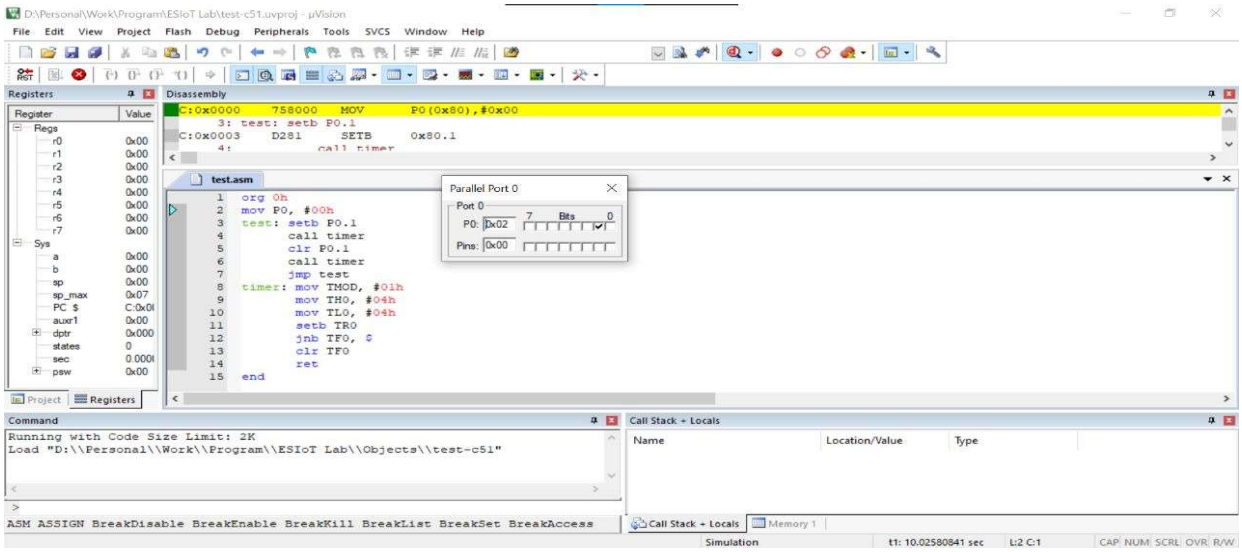
ALGORITHM:

1. Set the port 0 as an output pin
2. Set the value of pin1 of port 0 as high
3. Invoke the timer function to generate delay
4. Set the value of pin1 of port 0 as low
5. Invoke the timer function to generate delay
6. Repeat steps 2-5 infinitely
7. In the timer function, set the value of TMOD to use Mode 1 of Timer 0
8. Set the initial value for the timer in TH0 and TL0
9. Start the timer by setting TR0 to 1
10. Watch for rollover by checking TF0
11. If TF0 becomes 1, stop timer by setting TR0 to 0 and reset flag TF0

CODE:

```
org 0h
mov P0, #00h
test:
    setb P0.1
    call timer
    clr P0.1
    call timer
    sjmp test
timer:
    mov TMOD, #01h
    mov TH0, #04h
    mov TL0, #04h
    setb TR0
    jnb TF0, $
    clr TR0
    ret
end
```

OUTPUT:



2.Timer Delay program with interrupts:

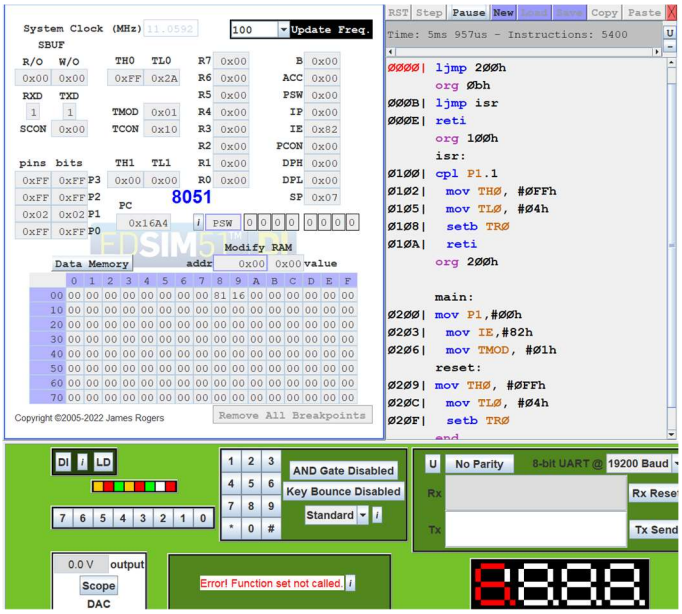
ALGORITHM:

1. Set the port 0 as an output pin
2. Set the value of pin1 of port 0 as high
3. Invoke the timer function to generate delay
4. Set the value of pin1 of port 0 as low
5. Invoke the timer function to generate delay
6. Repeat steps 2-5 infinitely
7. In the timer function, set the value of TMOD to use Mode 1 of Timer 0
8. Set the initial value for the timer in TH0 and TL0
9. Start the timer by setting TR0 to 1
10. Watch for rollover by checking TF0
11. If TF0 becomes 1, stop timer by setting TR0 to 0 and reset flag TF0

CODE:

```
org 0h
ljmp 200h
org 0bh
ljmp isr
reti
org 100h
isr:
cpl P1.1
      mov TH0, #0FFh
      mov TL0, #04h
      setb TR0
      reti
org 200h
main:
mov P1,#00h
      mov IE,#82h
      mov TMOD, #01h
reset:
mov TH0, #0FFh
      mov TL0, #04h
      setb TR0
end
```

OUTPUT:



RESULT:

Thus, the assembly language program to generate a time delay has been written and verified.

Date: 17/03/23 **Exp. No.:** 09	**BLINKING A LED**

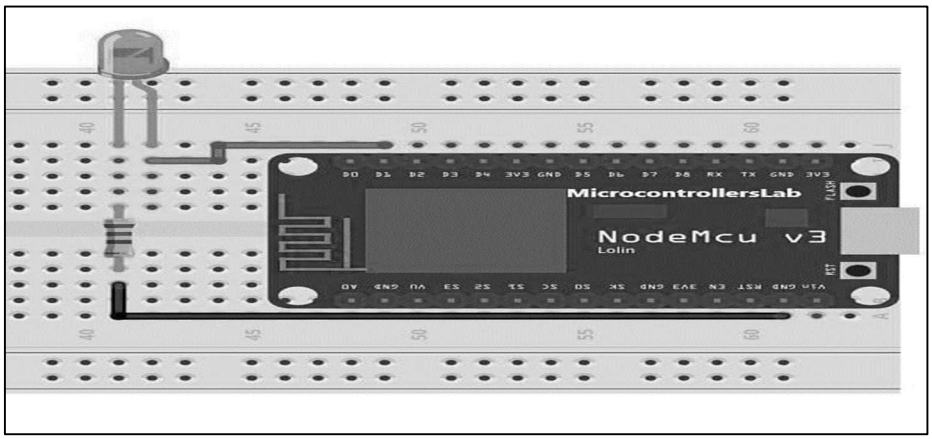
AIM:

To blink a LED using Node MCU.

HARDWARE REQUIRED:

- NodeMCU x 1
- LED x 1
- Breadboard
- 200 ohm – 1K ohm resistor x 1
- Micro USB cable x 1
- PC x 1
- Software Arduino IDE (version 1.6.4+)
- Jumper Wires (Male – Female & Male – Male)

CIRCUIT DIAGRAM:

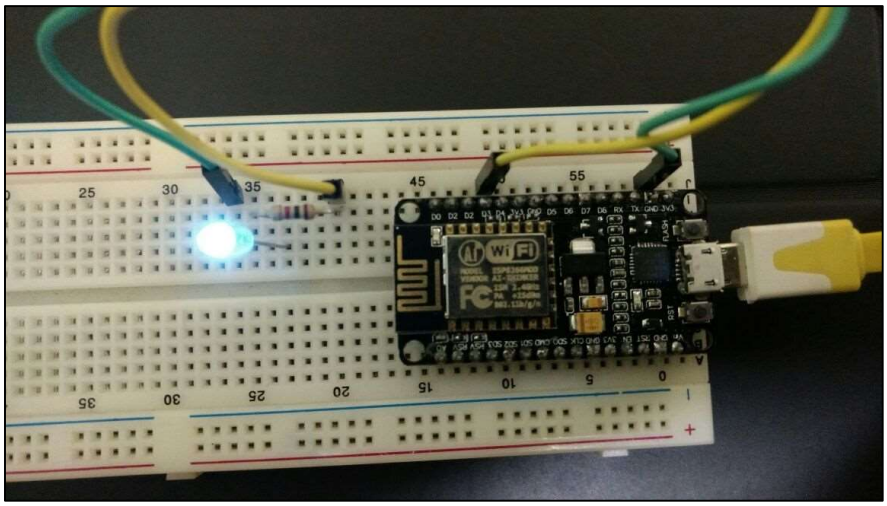


CODE:

```
int LED = 5;                        // Assign LED pin i.e: D1 on NodeMCU
void setup() {
    pinMode(LED, OUTPUT);           // initialize GPIO 5 as an output
}

void loop() {
    digitalWrite(LED, HIGH);        // turn the LED on
    delay(1000);                    // wait for a second
    digitalWrite(LED, LOW);         // turn the LED off
    delay(1000);                    // wait for a second
}
```

OUTPUT:



RESULT:

Thus, the blinking of LED has been done successfully using Node MCU and the results are verified.

Date: 17/03/23 **Exp. No.:** 10	**MEASURING TEMPERATURE USING TEMPERATURE SENSOR**

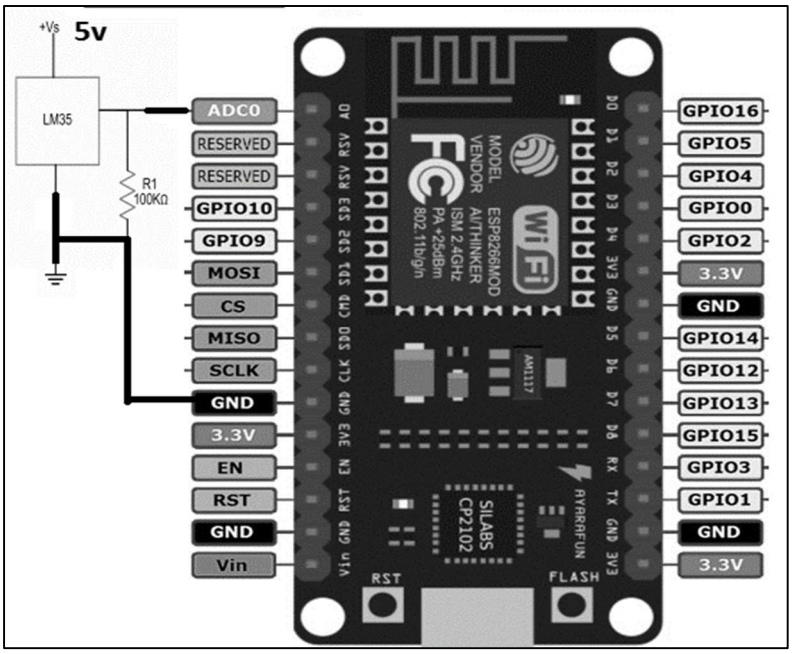
AIM:

To measure the temperature of the room / surrounding using temperature sensor & Node MCU.

HARDWARE REQUIRED:

- NodeMCU
- LM35 Temperature Sensor
- Bread Board
- Jumper Wires
- Micro USB Cable
- Arduino IDE
- 200 ohm – 1K ohm resistor x 1
- PC

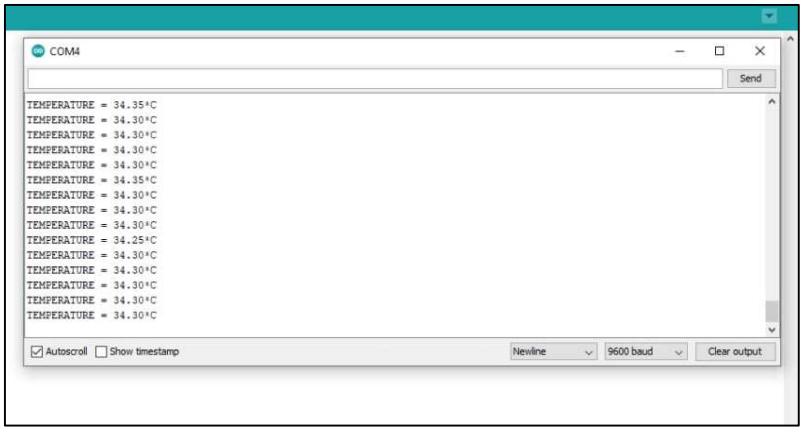
CIRCUIT DIAGRAM:



CODE:

```
int outputpin= A0;
void setup() {
    Serial.begin(9600);
}
void loop() {
    int analogValue = analogRead(outputpin);
    float millivolts = (analogValue/1024.0) * 3300;
    float celsius = millivolts/10;
    Serial.print("TEMPERATURE =   ");
    Serial.println(celsius);
    delay(1000);
}
```

OUTPUT:



RESULT:

Thus, the temperature has been measured successfully using Temperature sensor and the results are verified.

Date: 24/03/23 **Exp. No.:** 11	**INFRARED SENSOR USING NODE MCU**

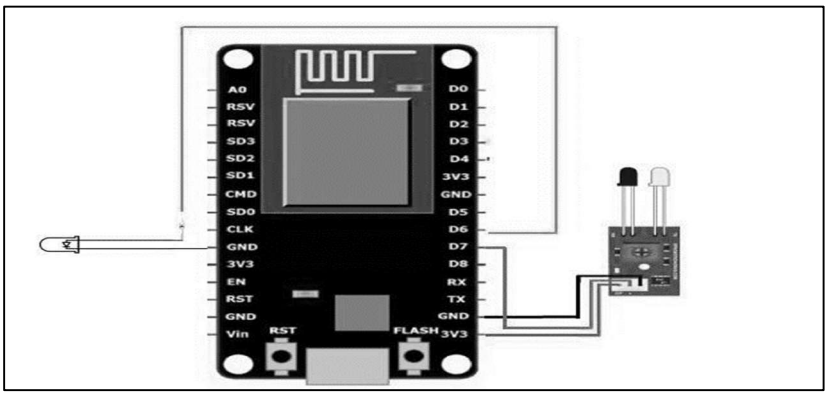
AIM:

To detect any motion in the room / surrounding using temperature sensor & Node MCU.

HARDWARE REQUIRED:

- NodeMCU
- IR Sensor
- Bread Board
- Jumper Wires
- Micro USB Cable
- Arduino IDE
- 200 ohm – 1K ohm resistor x 1
- PC
- LED

CIRCUIT DIAGRAM:



CODE:

```
int ledPin = 12;                    // choose pin for the LED
int inputPin = 13;                  // choose input pin (for Infrared sensor)
int val = 0;                        // variable for reading the pin status
void setup() {
   pinMode(ledPin, OUTPUT);         // declare LED as output
   pinMode(inputPin, INPUT);        // declare Infrared sensor as input
}
void loop(){
   val = digitalRead(inputPin);     // read input value
   if (val == HIGH)                 // check if the input is HIGH
         digitalWrite(ledPin, HIGH); // turn LED OFF
   else  digitalWrite(ledPin, LOW);  // turn LED ON
}
```

OUTPUT:

IR sensor identifies object nearby and turns LED ON
If no object is present in front, then LED is OFF.

RESULT:

Thus, IR sensor has been used successfully using Node MCU and the results are verified.

Date: 24/03/23 **Exp. No.:** 12	**LIGHT SENSOR**

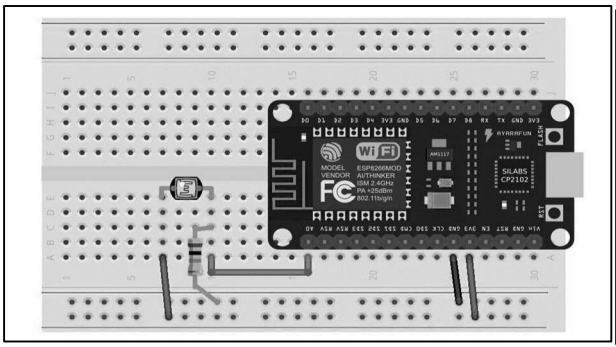
AIM:

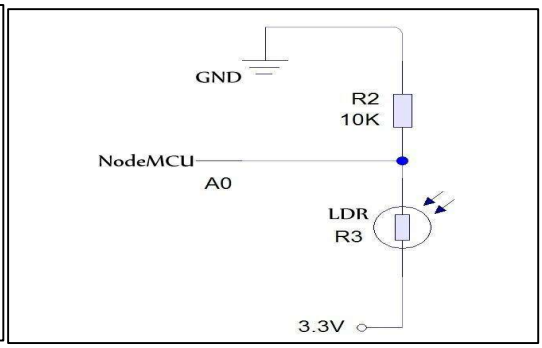
To use Light Sensor using Node MCU.

HARDWARE REQUIRED:

- NodeMCU
- LDR/PhotoResistor
- Bread Board
- Jumper Wires
- Micro USB Cable
- Arduino IDE (with ESP8266 Library installed)
- 200 ohm – 1K ohm resistor x 1
- PC

CIRCUIT DIAGRAM:



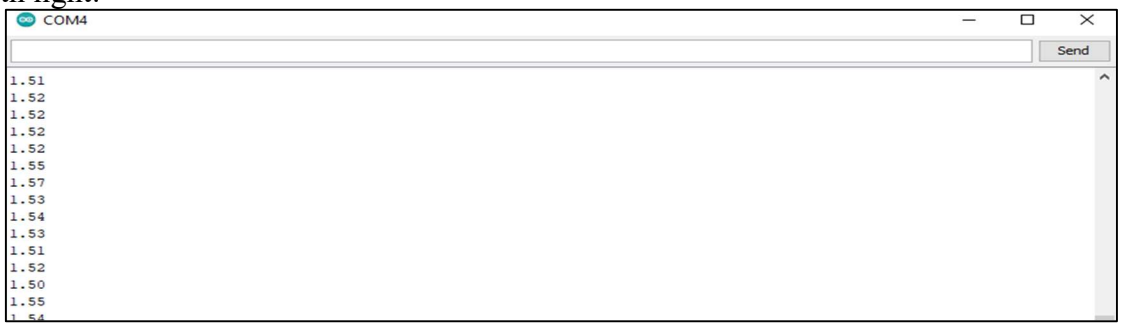


CODE:

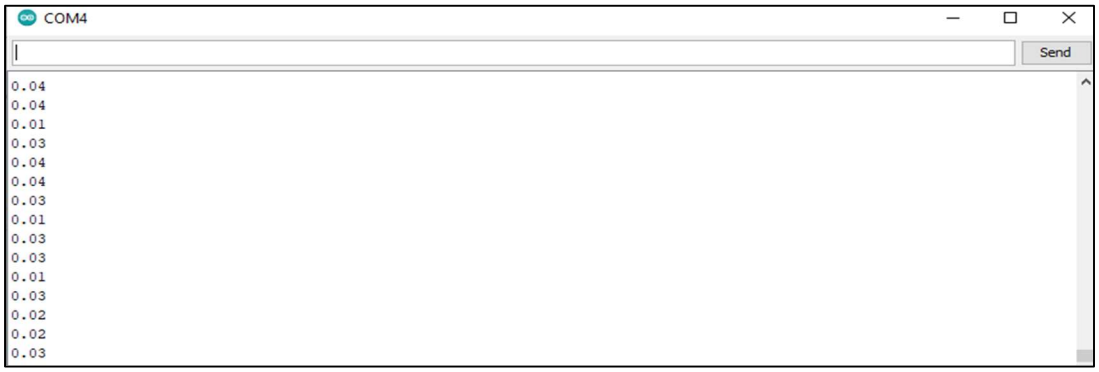
```
void setup() {
        Serial.begin(9600);
}
void loop() {
        int sensorValue = analogRead(A0);                    // read the input on analog pin 0
        float voltage = sensorValue * (5.0 / 1023.0);
        // Convert the analog reading (which goes from 0 - 1023) to a voltage (0 - 5V)
        Serial.println(voltage);                             // print out the value you read
}
```

OUTPUT:

With light:



Without light:



RESULT:

Thus, light sensor has been used successfully using Node MCU and the results are verified.

Date: 31/03/23 **Exp. No.:** 13	**PROXIMITY DETECTION**

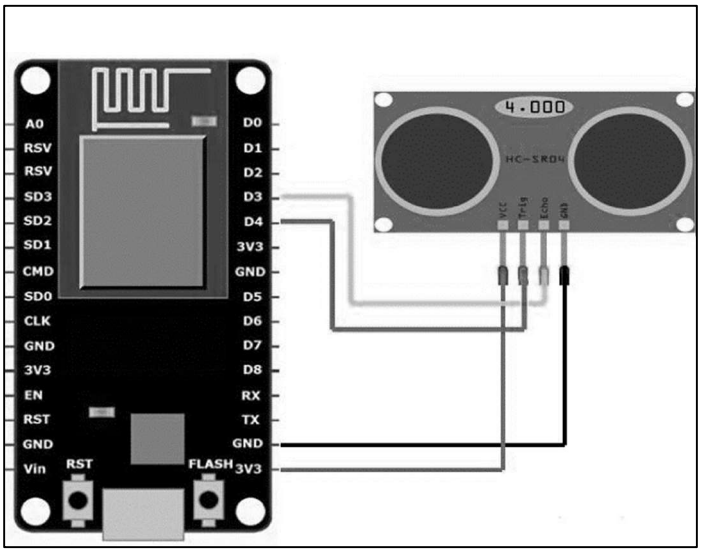
AIM:

To detect the location using ultrasonic sensor.

HARDWARE REQUIRED:

- NodeMCU
- HC-SR04 (Ultra-sonic Sensor)
- Bread Board
- Jumper Wires
- Micro USB Cable

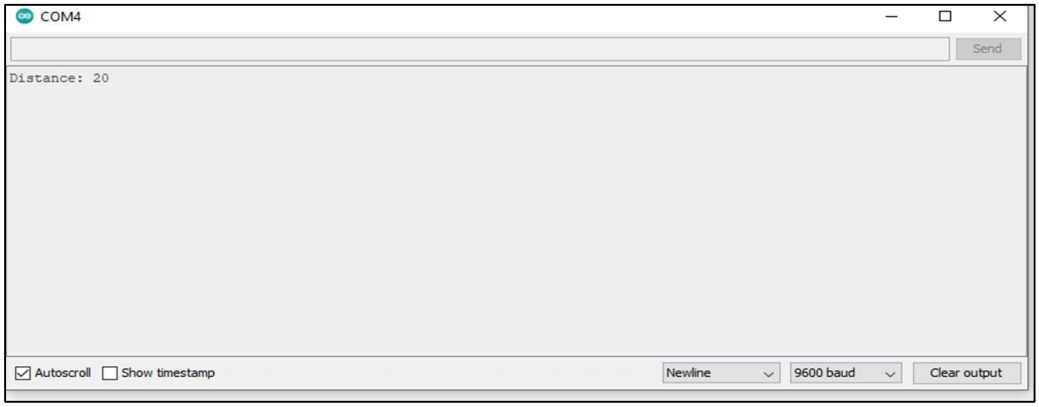
CIRCUIT DIAGRAM:



CODE:

```
void loop()  {
        digitalWrite(D6,LOW);                    //trigger sonic waves
        delayMicroseconds(2);
        digitalWrite(D6,HIGH);
        delayMicroseconds(10);
        long duration = pulseIn(D7,HIGH);        // receive ECHO and find DISTANCE
        float dist = duration * 0.034/2;
        Serial.println("Distance is...");
        Serial.println(dist);
        delay(2000);
}
```

OUTPUT:



RESULT:

Thus, proximity detection using ultra sonic sensor has been used successfully using Node MCU and the results are verified.

Date: 31/03/23 **Exp. No.:** 14	**SERVO MOTOR**

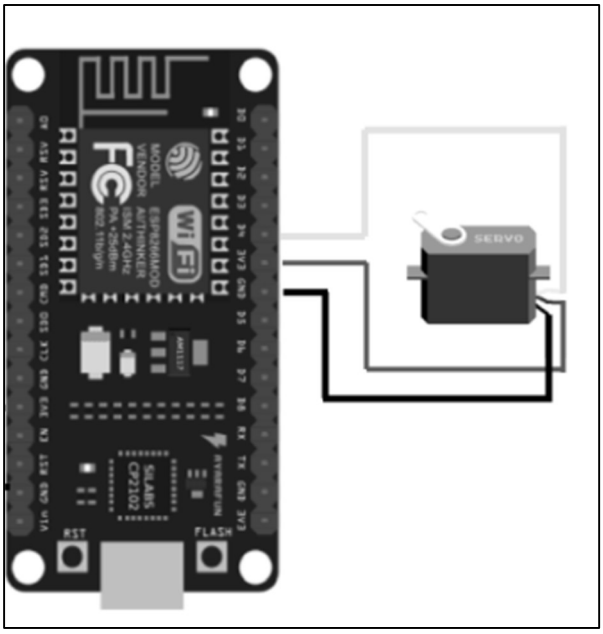
AIM:

To push or rotate an object with great precision at specific angles or distance using servo motor.

HARDWARE REQUIRED:

- NodeMCU
- Servo Motor
- Bread Board
- Jumper Wires
- Micro USB Cable

CIRCUIT DIAGRAM:



CODE:

```
#include <Servo.h>
Servo newservo1;
void setup() {
        newservo1.attach(D4);
}
 void loop() {
        newservo1.write(180);
        delay(1000);
        newservo1.write(0);
        delay(1000);
}
```

OUTPUT:

Servo Motor rotates 180 degree and comes back with 1 second delay.

RESULT:

Thus, servo motor has been successfully rotated using Node MCU and the results are verified.

Date: 21/04/23 **Exp. No.:** 15	**POTENTIOMETER WITH SERVO MOTOR**

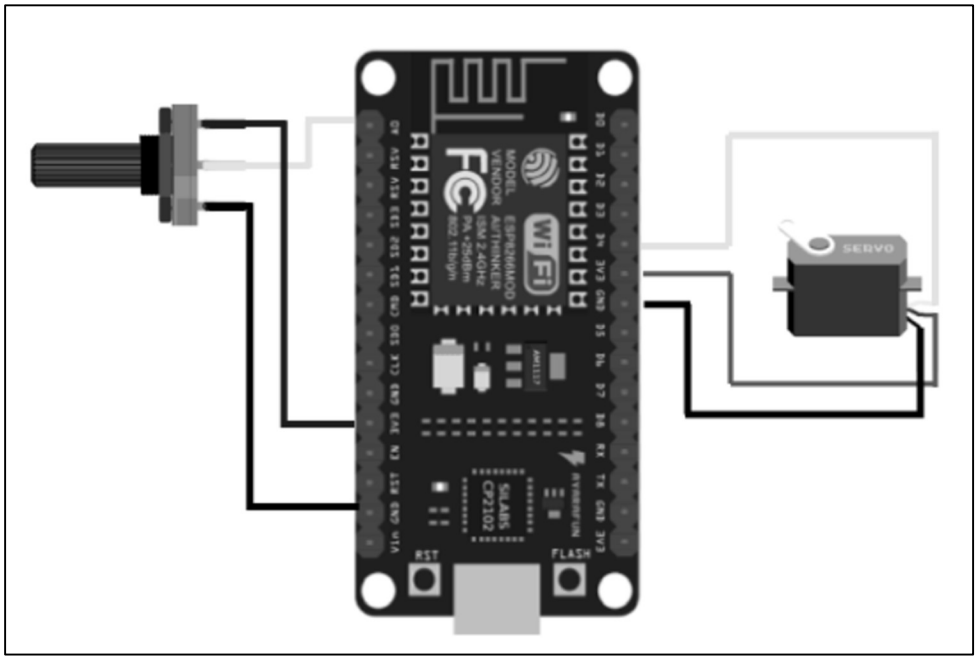
AIM:

To use a potentiometer with servo motor.

HARDWARE REQUIRED:

- NodeMCU
- Servo motor
- 10K POT
- Bread Board
- Jumper Wires
- Micro USB Cable

CIRCUIT DIAGRAM:



CODE:

```
#include <Servo.h>
#define CLK D3
#define DT D2
Servo servo;
int servoAngle=0
int prevCLK;
void setup() {
        pinMode(CLK,INPUT);
        pinMode(DT,INPUT);
        servo.attach(D4);
        prevCLK=digitalRead(CLK);
}
void loop() {
        crntCLK=digitalRead(CLK);
```

```
        if(crntCLK!=prevCLK){
            if(digitalRead(DT)==LOW && crntCLK==LOW) {
                    ServoAngle - =20;
                    if(servoAngle<0)
                            servoAngle=0;
            }
            else if(digitalRead(DT)==LOW) {
                    ServoAngle + =20;
                    if(servoAngle>180)
                            servoAngle=180;
            }
        }
        lastCLK=crntCLK;
    }
```

OUTPUT:

Servo Motor rotates based on the input to rotary encoder.

RESULT:

Thus, servo motor with potentiometer has been successfully rotated using Node MCU and the results are verified.

Date: 21/04/23 **Exp. No.:** 16	**IOT WITH CLOUD**

AIM:

To connect with Internet through Wi-Fi Access point using Node MCU.

HARDWARE REQUIRED:

- NodeMCU
- Bread Board
- Jumper Wires
- Micro USB Cable

CODE:

```
#include<ESP8266WiFi.h>
char* SSid = “Moto G82 5G”;
char*  Password = “password”
void setup() {
        WiFi.begin(SSid, Password);
        Serial.begin(115200);
        Serial.print(“Connecting: ”);
        while (WiFi.status()!= WL_CONNECTED) {
            Serial.print(“Waiting to connect”)
            delay(1000);
        }
        Serial.println(‘\n’);
        Serial.println (“Connection established”);
        Serial.println (“IP Address\t”);
        Serial.println(WiFi.LocalIP());
}
```

OUTPUT:

Devices connected:	1 of 8	
Device name	IP address	Physical address (MAC)
ESP-6DC72E	192.168.137.180	c4:5b:be:6d:c7:2e

```
C:\Users\student>ping 192.168.137.180

Pinging 192.168.137.180 with 32 bytes of data:
Reply from 192.168.137.180: bytes=32 time=4ms TTL=255
Reply from 192.168.137.180: bytes=32 time=3ms TTL=255
Reply from 192.168.137.180: bytes=32 time=2ms TTL=255
Reply from 192.168.137.180: bytes=32 time=1ms TTL=255

Ping statistics for 192.168.137.180:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 1ms, Maximum = 4ms, Average = 2ms

C:\Users\student>
```

RESULT:

Thus, connecting with internet has been done successfully using Node MCU and the results are verified.

Date: 28/04/23 **Exp. No.:** 17	**MAKING A NODE SERVER**

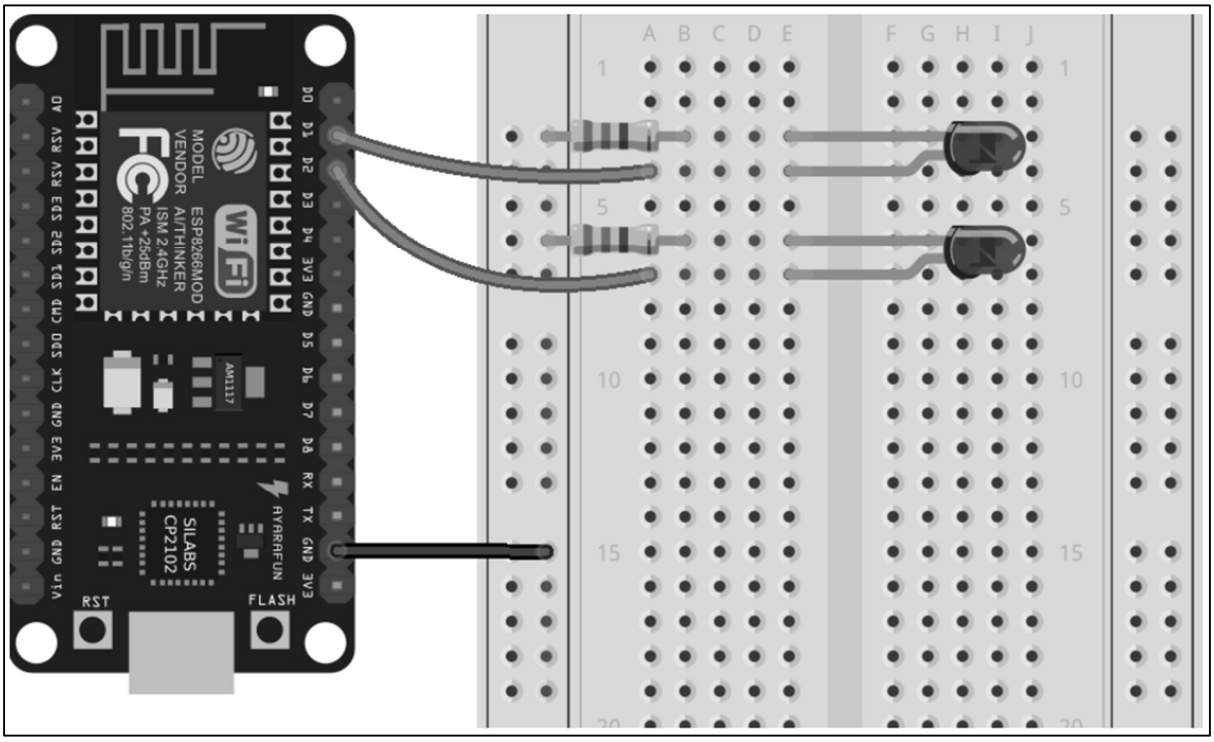
AIM:

To make a node as a server using Node MCU.

HARDWARE REQUIRED:

- NodeMCU
- LED
- Jumper Wires
- Micro USB Cable

CIRCUIT DIAGRAM:



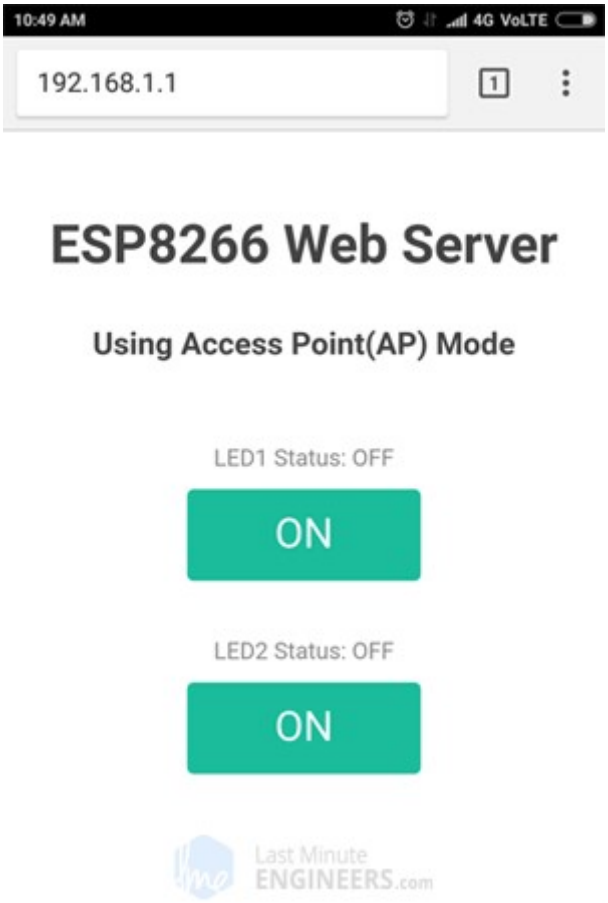
CODE:

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
const char* ssid = "Moto G82 5G";  // Enter SSID here
const char* password = "password";  //Enter Password here
IPAddress local_ip(192,168,1,1);
IPAddress gateway(192,168,1,1);
IPAddress subnet(255,255,255,0);
ESP8266WebServer server(80);
uint8_t LED1pin = D7;
bool LED1status = LOW;
uint8_t LED2pin = D6;
bool LED2status = LOW;
void setup() {
  Serial.begin(115200);
```

```
  pinMode(LED1pin, OUTPUT);
  pinMode(LED2pin, OUTPUT);
  WiFi.softAP(ssid, password);
  WiFi.softAPConfig(local_ip, gateway, subnet);
  delay(100);
  server.on("/", handle_OnConnect);
  server.on("/led1on", handle_led1on);
  server.on("/led1off", handle_led1off);
  server.on("/led2on", handle_led2on);
  server.on("/led2off", handle_led2off);
  server.onNotFound(handle_NotFound);
  server.begin();
  Serial.println("HTTP server started");
}
void loop() {
  server.handleClient();
  if(LED1status) digitalWrite(LED1pin, HIGH);
  else digitalWrite(LED1pin, LOW);
  if(LED2status) digitalWrite(LED2pin, HIGH);
  else digitalWrite(LED2pin, LOW);
}
void handle_OnConnect() {
  LED1status = LOW;
  LED2status = LOW;
  Serial.println("GPIO7 Status: OFF | GPIO6 Status: OFF");
  server.send(200, "text/html", SendHTML(LED1status,LED2status));
}
void handle_led1on() {
  LED1status = HIGH;
  Serial.println("GPIO7 Status: ON");
  server.send(200, "text/html", SendHTML(true,LED2status));
}
void handle_led1off() {
  LED1status = LOW;
  Serial.println("GPIO7 Status: OFF");
  server.send(200, "text/html", SendHTML(false,LED2status));
}
void handle_led2on() {
  LED2status = HIGH;
  Serial.println("GPIO6 Status: ON");
  server.send(200, "text/html", SendHTML(LED1status,true));
}
void handle_led2off() {
  LED2status = LOW;
  Serial.println("GPIO6 Status: OFF");
  server.send(200, "text/html", SendHTML(LED1status,false));
}
void handle_NotFound(){
  server.send(404, "text/plain", "Not found");
}
String SendHTML(uint8_t led1stat,uint8_t led2stat){
  String ptr = "<!DOCTYPE html> <html>\n";
  ptr +="<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-
      scalable=no\">\n";
```

```
    ptr +="<title>LED Control</title>\n";
    ptr +="</head>\n";
    ptr +="<body>\n";
    ptr +="<h1>ESP8266 Web Server</h1>\n";
    ptr +="<h3>Using Access Point(AP) Mode</h3>\n";
     if(led1stat) ptr +="<p>LED1 Status: ON</p><a class=\"button button-off\"
        href=\"/led1off\">OFF</a>\n";
    else ptr +="<p>LED1 Status: OFF</p><a class=\"button button-on\" href=\"/led1on\">ON</a>\n";
    if(led2stat) ptr +="<p>LED2 Status: ON</p><a class=\"button button-off\"
        href=\"/led2off\">OFF</a>\n";
    else ptr +="<p>LED2 Status: OFF</p><a class=\"button button-on\" href=\"/led2on\">ON</a>\n";
    ptr +="</body>\n";
    ptr +="</html>\n";
    return ptr;
  }
```

OUTPUT:



RESULT:

Thus, making a node has been done successfully using Node MCU and the results are verified.

Date: 19/05/23 **Exp. No.:** 18	**COLLECTING AND PROCESSING DATA FROM IOT SYSTEMS IN THE CLOUD USING THINGSPEAK**

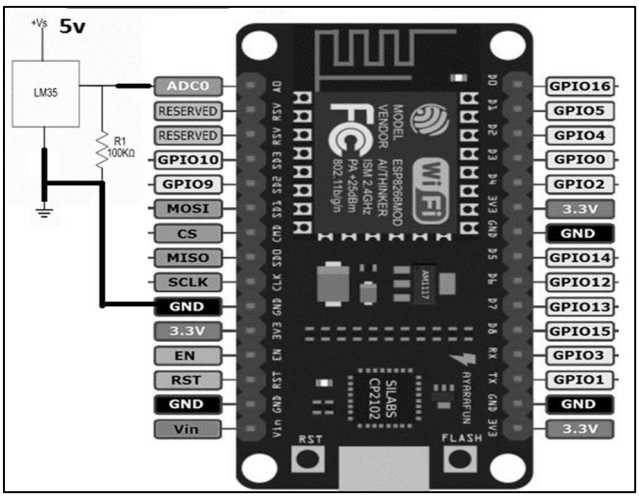
AIM:

To collecting and processing data from IoT systems in the cloud using Thing Speak.

HARDWARE REQUIRED:

- Node MCU
- Jumper Wire
- LM35 Temperature Sensor
- Micro USB Cable
- Breadboard

CIRCUIT DIAGRAM:

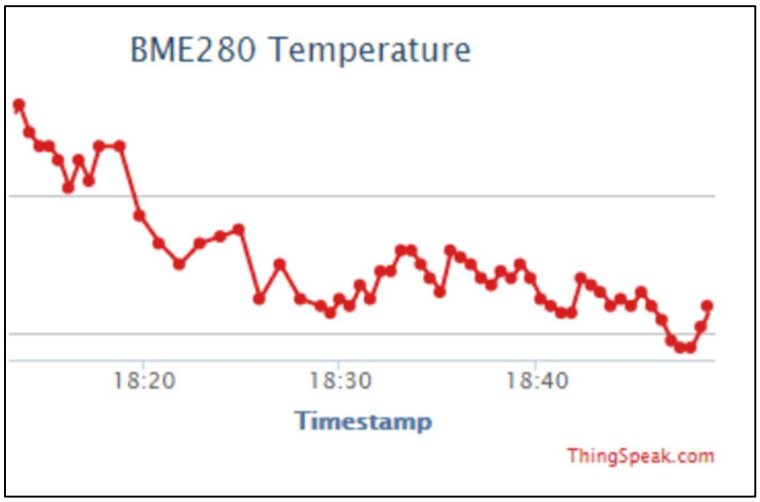


CODE:

```
#include "ThingSpeak.h"
#include <ESP8266WiFi.h>
char msg[50];
const char *ssid = "phone";    // your network SSID (name)
const char *password = "pass"; // your network password
WiFiClient client;
unsigned long myChannelNumber = 0000000;   // Your channel number
const char *myWriteAPIKey = "xxxxxxxxxxx"; // Your WriteAPI Key
// Timer variables
unsigned long lastTime = 0;
unsigned long timerDelay = 30000;
// Variable to hold temperature readings
float temperatureC;
int outputpin = A0;
void setup() {
  Serial.begin(115200); // Initialize serial
  WiFi.mode(WIFI_STA);
  ThingSpeak.begin(client); // Initialize ThingSpeak
  if (WiFi.status() != WL_CONNECTED) {
    Serial.print("Attempting to connect");
```

```
        while (WiFi.status() != WL_CONNECTED) {
          WiFi.begin(ssid, password);
          delay(5000);
        }
        Serial.println("\nConnected.");
      }
    }
    void loop() {
      if ((millis() - lastTime) > timerDelay || 1) {
        int analogValue = analogRead(outputpin);
        float millivolts = (analogValue / 1024.0) * 3300;
        temperatureC = millivolts / 80 + random(10);
        Serial.print("Temperature (ºC): ");
        Serial.println(temperatureC);
        int x =
           ThingSpeak.writeField(myChannelNumber, 1, temperatureC, myWriteAPIKey);
        if (x == 200) {
          Serial.println("Channel update successful.");
        } else {
          Serial.println("Problem updating channel. HTTP error code " + String(x));
        }
        lastTime = millis();
      }
    }
```

OUTPUT:



RESULT:

Thus, data has been successfully connected and processed using ThingSpeak.

Date: 19/05/23 **Exp. No.:** 19	**DEVELOP IOT APPLICATIONS USING BLUEMIX**

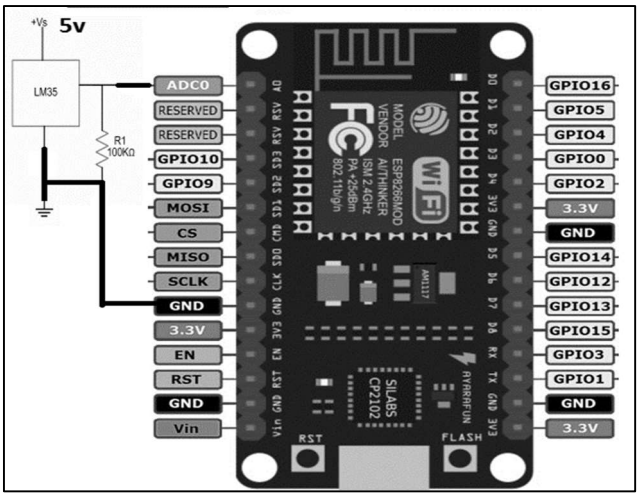
AIM:

To develop IoT applications using Django Framework and Firebase/ Bluemix platform.

HARDWARE REQUIRED:

- Node MCU
- Jumper Wire
- LM35 Temperature Sensor
- Micro USB Cable
- Breadboard

CIRCUIT DIAGRAM:



CODE:

```
#include <ArduinoJson.h>
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#define MQTT_HOST "vzrica.messaging.internetofthings.ibmcloud.com"
// change  su1efs
#define MQTT_PORT 1883
#define MQTT_DEVICEID "d:vzrica:ESP8266:dev01"
#define MQTT_USER "use-token-auth"
#define MQTT_TOKEN "Manoj_Selvam" // change your auth_id :
#define MQTT_TOPIC "iot-2/evt/status/fmt/json"
#define MQTT_TOPIC_DISPLAY "iot-2/cmd/display/fmt/json"
// Add WiFi connection information
char ssid[] = "phone"; //  your network SSID (name)
char pass[] = "pass";  // your network password
// MQTT objects
void callback(char *topic, byte *payload, unsigned int length);
WiFiClient wifiClient;
PubSubClient mqtt(MQTT_HOST, MQTT_PORT, callback, wifiClient);
// variables to hold data
StaticJsonDocument<100> jsonDoc;
JsonObject payload = jsonDoc.to<JsonObject>();
JsonObject status = payload.createNestedObject("d");
```

```
static char msg[50];
float t = 0.0;
void callback(char *topic, byte *payload,
            unsigned int length) { // handle message arrived
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] : ");
  payload[length] = 0;
  Serial.println((char *)payload);
}
void setup() {
  Serial.begin(115200);
  Serial.setTimeout(2000);
  while (!Serial) {
  }
  Serial.println();
  Serial.println("ESP8266 Sensor Application");
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi Connected");
  // Connect to MQTT - IBM Watson IoT Platform
  if (mqtt.connect(MQTT_DEVICEID, MQTT_USER, MQTT_TOKEN)) {
    Serial.println("MQTT Connected");
    mqtt.subscribe(MQTT_TOPIC_DISPLAY);
  } else {
    Serial.println("MQTT Failed to connect!");
    ESP.reset();
  }
}
void loop() {
  mqtt.loop();
  while (!mqtt.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
    if (mqtt.connect(MQTT_DEVICEID, MQTT_USER, MQTT_TOKEN)) {
      Serial.println("MQTT Connected");
      mqtt.subscribe(MQTT_TOPIC_DISPLAY);
      mqtt.loop();
    } else {
      Serial.println("MQTT Failed to connect!");
      delay(5000);
    }
  }
  // get t from temp sensor
  int analogValue = analogRead(A0);
  float millivolts = (analogValue / 1024.0) * 3300;
  t = millivolts / 40;
  if (isnan(t))
    Serial.println("Failed to read from DHT sensor!");
```

```
    else {
      status["temp"] = t + random(10);
      serializeJson(jsonDoc, msg, 50);
      Serial.println(msg);
      if (!mqtt.publish(MQTT_TOPIC, msg))
        Serial.println("MQTT Publish failed");
    }
    // Pause - but keep polling MQTT for incoming messages
    for (int i = 0; i < 10; i++) {
      mqtt.loop();
      delay(1000);
    }
  }
```

OUTPUT:

RESULT:

Thus, IoT applications using Bluemix platform has been successfully developed and the results are verified.