SRI KRISHNA COLLEGE OF TECHNOLOGY

[An Autonomous Institution | Affiliated to Anna University and Approved by AICTE | Accredited by NBA &NAAC with 'A' Grade]

KOVAIPUDUR, COIMBATORE – 641 042.

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

[Accredited by NBA]

ACADEMIC YEAR: 2024 – 2025 (ODD)

LAB MANUAL

22CS503 –MICROPROCESSORS AND MICROCONTROLLERS LABORATORY

(For III Year CSE - V Semester)

Prepared by

1. **Dr.P.Rajasekar**
2. **Dr.P.Jayarajan**
3. **Dr.M.Priyatharishini**
4. **Dr. K.Bagyalakshmi**
5. **Dr.N.Kirthika**

Verified By
Dr Senthilkumar C
Lab Co-ordinator

Approved By
Dr.K.Muthulakshmi
Prof & Head/ECE

SRI KRISHNA COLLEGE OF TECHNOLOGY
[An Autonomous Institution | Affiliated to Anna University and
Approved by AICTE | Accredited by NBA &NAAC with 'A' Grade]
KOVAIPUDUR, COIMBATORE – 641 042.

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

VISION

To produce technical manpower of global standards in Electronics and Communication Engineering with capabilities of adapting to new challenges to address the societal needs.

MISSION

M1: Equipping faculty members with knowledge in cutting-edge technologies through various programs.

M2: Imparting quality education to meet the requirements of all stakeholders with the help of well-qualified and experienced faculty resources.

M3: Nurturing competent professionals through extra and co-curricular activities.

M4: To promote research and development activities by setting up new research facilities and industrial interaction.

M5: Accomplishing the technological needs of the society.

PROGRAM EDUCTIONAL OBJECTIVES

Graduates of Electronics and Communication Engineering will be able to

PEO1: Apply the broad fundamental concepts of mathematics and science in Electronics and Communication Engineering to be competent enough in their chosen careers.

PEO2: Innovatively design, simulate, develop, implement and test hardware and software components for offering solution to real life problems and also to demonstrate effective communication skills, the ability to work well either individually or as part of a team.

PEO3: Become scientists, technocrats and business leaders by utilizing an intellectual environment through Centres of Excellence in research and development oriented emerging technologies and intensive training.

PEO4: Be a good human being and responsible citizen for the overall welfare of the society.

\

PROGRAM OUTCOMES

Graduates of Electronics and Communication Engineering program of Sri Krishna College of Technology will have the ability to

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects

and in multidisciplinary environments.

PO12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES

Graduates of Electronics and Communication Engineering program of Sri Krishna College of Technology will have the ability to

PSO1: An ability to understand the fundamental concepts and apply them to analyze, design, simulate and implement Electronic Circuits and Communication systems.

PSO2: An ability to offer real time solutions for problems related to Electronics and Communication Engineering and contribute towards the welfare of society.

22CS503	MICROPROCESSORS AND MICROCONTROLLERS LABORATORY	0/0/3/1.5

Nature of Course	Practical
Course Pre-requisites	NIL

Course Objectives:

1	To understand the basic programming of Microprocessors and Micro Controllers
2	To solve various real time problems using Micro controllers and Micro - processors.

Course Outcomes

Upon completion of the course, students shall have ability to

CO1	Write the assembly language programming using 8086 Microprocessor	[AP]
CO2	Develop program using 8051 microcontroller and Interface 8051 with other devices.	[AP]
CO3	Solve real time problems using Arduino Processor	[AP]
CO4	Develop program using ARM and to do interfacing with external circuit.	[AP]
CO5	Design solutions for real time problems using Arduino Processor	[AP]

Course Contents

S.No.,	List of Experiments	CO Mapping	RBT
1.	Assembly Language programs using 8086.	CO1	[AP]
2.	Assembly Language programs using 8051.	CO2	[AP]
3.	Stepper motor control using 8086 Microprocessor.	CO1	[AP]
4.	Sensor Interfacing using 8051 Micro controller.	CO2	[AP]
5.	Interfacing 8051 with ADC.	CO2	[AP]
6.	Basic Programming with Arduino Kit	CO3	[AP]
7.	Design of a Traffic light controller with Arduino.	CO3	[AP]
8.	Design a Simple chat Server using Arduino.	CO3	[AP]
9.	Basic programming using ARM Processor.	CO4	[AP]
10.	Interfacing with seven segment display using ARM.	CO5	[AP]
11.	Design solutions for smart home using Arduino Processor	CO5	[AP]
		Total Hours	**45 Hrs**

Text Books:

1.	A.K.Ray & K.M.Bhurchandi, "Advanced Microprocessors and peripherals- Architectures, Programming and Interfacing", Third edition, TMH, 2012 Reprint.
2.	Mohamed Ali Mazidi, Janice GillispieMazidi, "The 8051 microcontroller and embedded systems", Pearson Education,2006.

Reference Books:

1	Andrew N.Sloss, Dominic Symes, Chris Wright "ARM System Developer"s Guide : Designing and Optimizing System Software" , First edition, Morgan Kaufmann Publishers, 2004.
2	Simon Monk "Programming Arduino getting started with sketches", The McGraw-Hill, 2012.

Web References:

1.	https://www.tutorialspoint.com/microprocessor/microprocessor_overview.htm
2.	https://www.tutorialspoint.com/microprocessor/microcontrollers_overview.htm

Online Resources:

1.	https://onlinecourses.nptel.ac.in/noc21_ee18/preview

2.	http://vlabs.iitb.ac.in/vlabs-dev/labs/8051-Microcontroller-Lab/labs/index.php
3.	http://vlabs.iitb.ac.in/vlabs-dev/labs_local/microprocessor/labs/explist.php

CO	PO												PSO	
	1	2	3	4	5	6	7	8	9	10	11	12	1	2
CO1	3	3	3	3	3	-	-	-	-	-	-	2	2	2
CO2	3	3	3	3	3	-	-	-	-	-	-	2	2	2
CO3	3	3	3	3	3	-	-	-	-	-	-	2	2	2
CO4	3	3	3	3	3	-	-	-	-	-	-	2	2	2
CO5	3	3	3	3	3	-	-	-	-	-	-	2	2	2

CO- Course Outcome
PO- Programme Outcome
PSO- Programme Specific Outcomes

1- Reasonably Agreed
2- Moderately Agreed
3- Strongly Agreed

SRI KRISHNA COLLEGE OF TECHNOLOGY
[An Autonomous Institution | Affiliated to Anna University and
Approved by AICTE | Accredited by NBA &NAAC with 'A' Grade]
KOVAIPUDUR, COIMBATORE – 641 042.

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

MICROPROCESSORS AND MICROCONTROLLERS LABORATORY

DO's

1. Follow proper dress code and be regular to lab.

2. Understand the theory behind the lab before coming to the lab.

3. Get the signature of the in-charge before getting the components.

4. Handle breadboard and components properly.

5. Keep the work area neatly.

6. After the completion of the experiments switch off the power supply and return the components.

7. Arrange the stools and equipment's properly before leaving the lab.

DON'TS

1. Don't exceed voltage rating and avoid loose connections and short circuits.

2. Avoid replacing the component while power supply is ON

3. Don't throw the connecting wires on the floor.

4. Don't use Mobile phones or electronic gadgets inside laboratory.

INDEX

INTRODUCTION TO MICROPROCESSORS AND MICROCONTROLLERS LAB

INTRODUCTION TO 8086 MICROPROCESSOR

Objective

To study the components, functionality and overall operations of 8086 microprocessor architecture.

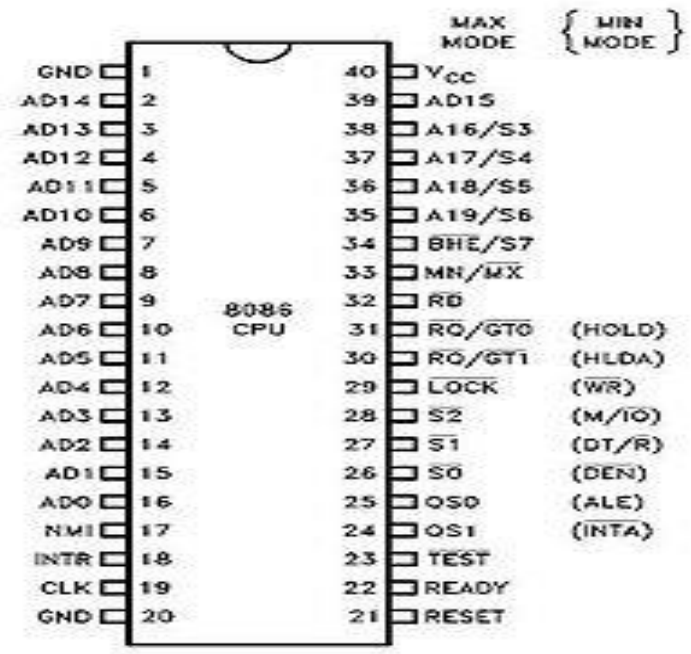
8086 MICROPROCESSOR

Features of 8086 microprocessor

- It is a 16 bit µp.
- 8086 has a 20 bit address bus can access up to 220 memory locations (1 MB) .
- It can support up to 64K I/O ports.
- It provides 14, 16-bit registers.
- It has multiplexed address and data bus AD0- AD15 and A16 – A19.
- It requires single phase clock with 33% duty cycle to provide internal timing.
- 8086 is designed to operate in two modes, Minimum and Maximum.
- It can prefetch up to 6 instruction bytes from memory and queues them in order to speed up instruction execution.
- It requires +5V power supply.

8086 Pin Diagram

- **AD15 - AD0** : These are the time multiplexed memory I/O address and data lines.
- **A19/S6, A18/S5,A17/S4, A16/S3**: These are the time multiplexed address and status lines.
- **BHE/S7** : The bus high enable is used to indicate the transfer of data over the higher order (D15-D8) data bus



- **RD** – **Read** : This signal on low indicates the peripheral that the processor is performing memory or I/O read operation
- **READY** : This is the acknowledgement from the slow device or memory that they have completed the data transfer
- **INTR-Interrupt Request:** This is a triggered input. This is sampled during the last clock cycles of each instruction to determine the availability of the request
- **TEST** : This input is examined by a 'WAIT' instruction
- **CLK**- Clock Input : The clock input provides the basic timing for processor operation and bus control activity
- **MN/MX:** The logic level at this pin decides whether the processor is to operate in either minimum or maximum mode.
- **M/IO – Memory/IO:** This is a status line logically equivalent to S2 in maximum mode. When it is low, it indicates the CPU is having an I/O operation, and when it is high, it indicates that the CPU is having a memory operation.

8086 Microprocessor Hardware Kit

8086 INSTRUCTION SET

OPCODE		DESCRIPTION
AAA		ASCII adjust addition
AAD		ASCII adjust division
AAM		ASCII adjust multiply
AAS		ASCII adjust subtraction
ADC	dt,sc	Add with carry
ADD	dt,sc	Add
AND	dt,sc	Logical AND
CALL	proc	Call a procedure
CBW		Convert byte to word
CLC		Clear carry flag
CDL		Clear direction flag
CLI		Clear interrupt flag
CMC		Complement carry flag
CMP	dt,sc	Compare
CMPS	[dt,sc]	Compare string
CMPSB	"	" bytes
CMPSW	"	" words
CWD		Convert word to double word
DAA		Decimal adjust addition
DAS		Decimal adjust subtraction
DEC	dt	Decrement
DIV	sc	Unsigned divide
ESC	code,sc	Escape
HLT		Halt
IDIV	sc	Integer divide
IMUL	sc	Integer multiply
IN	ac,port	Input from port
INC	dt	Increment
INT	type	Interrupt
INTO		Interrupt if overflow
IRET		Return from interrupt
JA	slabel	Jump if above
JAE	slabel	Jump if above or equal
JB	slabel	Jump if below
JBE	slabel	Jump if below or equal
JC	slabel	Jump if carry
JCXZ	slabel	Jump if CX is zero
JE	slabel	Jump if equal
JG	slabel	Jump if greater
JGE	slabel	Jump if greater or equal
JL	slabel	Jump if less
JLE	slabel	Jump if less or equal
JMP	label	Jump
JNA	slabel	Jump if not above
JNAE	slabel	Jump if not above or equal
JNB	slabel	Jump if not below
JNBE	slabel	Jump if below or equal
JNC	slabel	Jump if no carry
JNE	slabel	Jump if not equal
JNG	slabel	Jump if not greater
JNGE	slabel	Jump if not greater or equal
JNL	slabel	Jump if not less
JNLE	slabel	Jump if not less or equal
JNZ	slabel	Jump if not zero
JNO	slabel	Jump if not overflow
JNP	slabel	Jump if not parity
JNS	slabel	Jump if not sign
JO	slabel	Jump if overflow
JPO	slabel	Jump if parity odd
JP	slabel	Jump if parity
JPE	slabel	Jump if parity even
JS	slabel	Jump if sign
JZ	slabel	Jump if zero
LAHF		Load AH from flags
LDS	dt,sc	Load pointer using DS
LEA	dt,sc	Load effective address
LES	dt,sc	Load pointer using ES
LOCK		Lock bus
LODS	[sc]	Load string
LODSB	"	" bytes
LODSW	"	" words
LOOP	slabel	Loop
LOOPE	slabel	Loop if equal
LOOPZ	slabel	Loop if zero
LOOPNE	slabel	Loop if not equal
LOOPNZ	slabel	Loop if not zero
MOV	dt,sc	Move
MOVS	[dt,sc]	Move string
MOVSB	"	" bytes
MOVSW	"	" words
MUL	sc	Unsigned multiply
NEG	dt	Negate
NOP		No operation
NOT	dt	Logical NOT
OR	dt,sc	Logical OR
OUT	port,ac	output to port
POP	dt	Pop word off stack
POPF		Pop flags off stack
PUSH	sc	Push word onto stack
PUSHF		Push flags onto stack
RCL	dt,cnt	Rotate left through carry
RCR	dt,cnt	Rotate right through carry
REP		Repeat string operation
REPE		Repeat while equal
REPZ		Repeat while zero
REPNE		Repeat while not equal
REPNZ		Repeat while not zero
RET	[pop]	Return from procedure
ROL	dt,cnt	Rotate left
ROR	dt,cnt	Rotate right
SAHF		Store AH into flags
SAL	dt,cnt	Shift arithmetic left
SHL	dt,cnt	Shift logical left
SAR	dt,cnt	Shift arithmetic right
SBB	dt,sc	Subtract with borrow
SCAS	[dt]	Scan string
SCASB	"	" byte
SCASW	"	" word
SHR	dt,cnt	Shift logical right
STC		Set carry flag
STD		Set direction flag
STI		Set interrupt flag
STOS	[dt]	Store string
STOSB	"	" byte
STOSW	"	" word
SUB	dt,sc	Subtraction
TEST	dt,sc	Test (logical AND)
WAIT		Wait for 8087
XCHG	dt,sc	Exchange
XLAT	table	Translate
XLATB	"	"
XOR	dt,sc	Logical exclusive OR

Notes:
dt - destination
sc - source
label - may be near or far address
slabel - near address

Result:

Thus the functionality and the operations of the 8086 Microprocessor architecture were studied successfully.

Ex No: 1 a Date:	BASIC ARITHMETIC OPERATIONS USING 8086 MICROPROCESSORS

OBJECTIVE:

To develop 8086 arithmetic calculations for control processes such as temperature regulation, pressure control, and flow measurement in industrial environments using fundamental arithmetic operations, including addition, subtraction, multiplication, and division to manage operand sizes.

SYSTEM AND SOFTWARE TOOL REQUIRED:

1. 8086 Microprocessor kit
2. Keyboard

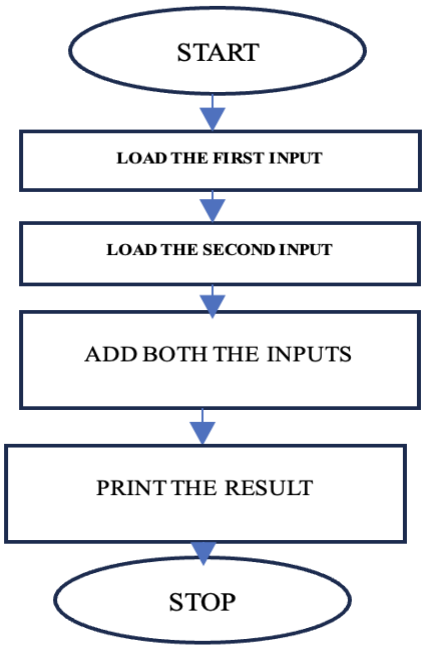
PRE LAB-WORK

ADDITION OF TWO NUMBERS

ALGORITHM:

1. Start the program
2. Load the 1st number to AX register.
3. Load the 2nd number to BX register.
4. Add both the numbers
5. Store the result in memory location
6. Halt the program.

FLOWCHART



IN LAB WORK

PROGRAM: ADDITION

Address	Mnemonics	Comments
1100	MOV AX, 8212H	Move the 1st number to AX register.
1103	MOV BX, 9313H	Move the 2nd number to AX register.
1106	ADD AX, BX	Add both the numbers
1108	MOV [1200], AX	Store the result to memory location.
110A	JC CARRY (1110)	Checks if the carry flag is set. If so, it jumps to the carry label
110D	JMP NOCARRY (1113)	Jumps to the NOCARRY label to avoid executing the carry
1110	CARRY: MOV [1201], 0001H	Store the value 0001H to indicate carry bit
1113	NOCARRY: MOV [1201],0000H	Store the value 0001H to indicate no carry
1116	HLT	Stop the program

POST LAB WORK

Input

Memory Location	Input Data
AX	8212H
BX	9313H

Output

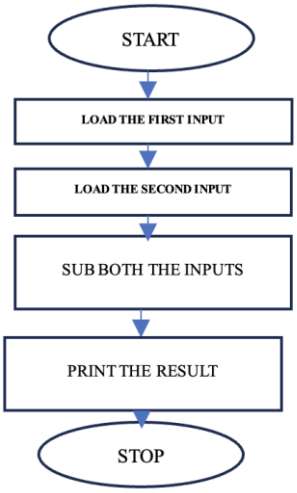
Memory Location	Output Data
1200	0525H
1201	0001H

SUBTRACTION OF TWO NUMBERS

ALGORITHM:

1. Start the program
2. Load the 1st number to AX register.
3. Load the 2nd number to BX register.
4. Subtract both the numbers
5. Store the result in memory location
6. Halt the program.

FLOWCHART



IN LAB WORK

PROGRAM : SUBTRACTION

Address	Mnemonics	Comments
1100	MOV AX, 1212H	Move the 1st number to AX register.
1103	MOV BX, 1313H	Move the 2nd number to AX register.
1106	SUB AX, BX	Subtract both the numbers
1108	MOV [1200], AX	Store the result to memory location.
101A	HLT	Stop the program

POST LAB WORK

Input

Memory Location	Input Data
AX	1212H
BX	1313H

Output

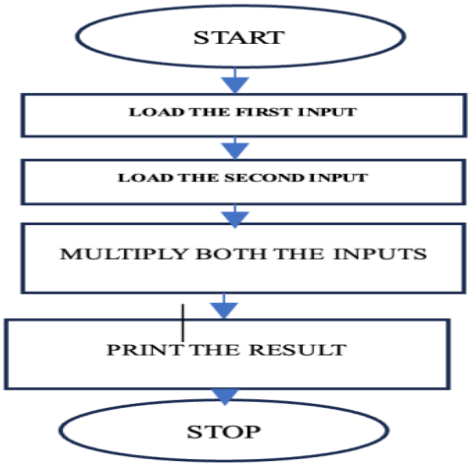
Memory Location	Output Data
1200	FEFFH

MULTIPLICATION OF TWO NUMBERS

ALGORITHM:

1. Start the program
2. Load the 1st number to AX register.
3. Load the 2nd number to BX register.
4. Multiply both the numbers
5. Store the result in memory location
6. Halt the program.

FLOWCHART



IN LAB WORK

PROGRAM: MULTIPLICATION

Address	Mnemonics	Comments
1100	MOV AX, 1212	Move the 1st number to AX register.
1103	MOV BX, 1313	Move the 2nd number to AX register.
1106	MUL AX, BX	Multiply the both the numbers
1108	MOV [1200], AX	Store the result to memory location.
101A	HLT	Stop the program

POST LAB WORK

Input

Memory Location	Input Data
AX	1212H
BX	1313H

Output

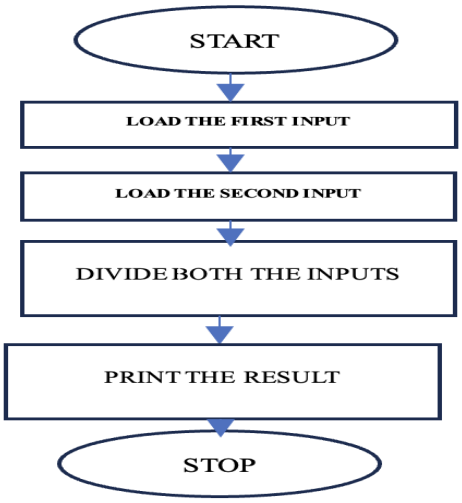
Memory Location	Output Data
1200	OE24H

DIVISION OF TWO NUMBERS

ALGORITHM:

1. Start the program
2. Load the 1st number to AX register.
3. Load the 2nd number to BX register.
4. Divide both the numbers
5. Store the result in memory location
6. Halt the program.

FLOWCHART



IN LAB WORK

PROGRAM: DIVISION

Address	Mnemonics	Comments
1100	MOV AX, 1000H	Move the 1st number to AX register.
1103	MOV BX, 0013H	Move the 2nd number to AX register.
1106	DIV AX, BX	Divide the both the numbers
1108	MOV [1200], AX	Store the result to memory location.
101A	HLT	Stop the program

POST LAB WORK

Input

Memory Location	Input Data
AX	1000H
BX	0013H

Output

Memory Location	Output Data
1200	00D7H

Application:

1. Arithmetic operations to process audio signals and video data in digital image processing
2. Motor Control Algorithms in robots and automated machinery.
3. Perform arithmetic calculations for control processes.
4. Financial Calculators.
5. Mathematical Modeling and Simulations.

Pre-Lab Viva Questions:

1. What are the main registers in the 8086 microprocessor?
2. What is clock speed?
3. What is logical address?
4. How do you define data in an 8086 assembly language program?
5. What is the difference between MUL and IMUL instructions in 8086?

Post-Lab Viva Questions:

1. How Data Copy/Transfer Instructions?
2. What is the purpose of Opcode?
3. _____ is an external signal that causes a microprocessor to jump to a specific subroutine.
4. Manually perform the 8 bit and 16 bit data addition process and also validate the result obtain from the microprocessor.
5. List the scenario where the carry flag would be set after an addition operation.

RESULT:

Thus, the 8086 arithmetic calculations for control processes such as temperature regulation, pressure control, and flow measurement in industrial environments using fundamental arithmetic operations, including addition, subtraction, multiplication, and division are calculated using 8086 microprocessor.

Ex No: 1 b Date:	ARRAY PROGRAMMING- FINDING LARGEST AND SMALLEST NUMBER

OBJECTIVE:

To develop a system to analyze student grades, by determining the highest and lowest grades in an array and providing a grade distribution using 8086 trainer kit.

SYSTEM AND SOFTWARE TOOL REQUIRED:

1. 8086 Microprocessor kit
2. Keyboard

PRE LAB WORK

ALGORITHM:

i) Finding Largest Number

Step1: Load the array count in a register C

Step2: Get the first two numbers

Step3: Compare the numbers and exchange if the number is small

Step4: Get the third number from the array and repeat the process until C is 0

ii) Finding smallest number:

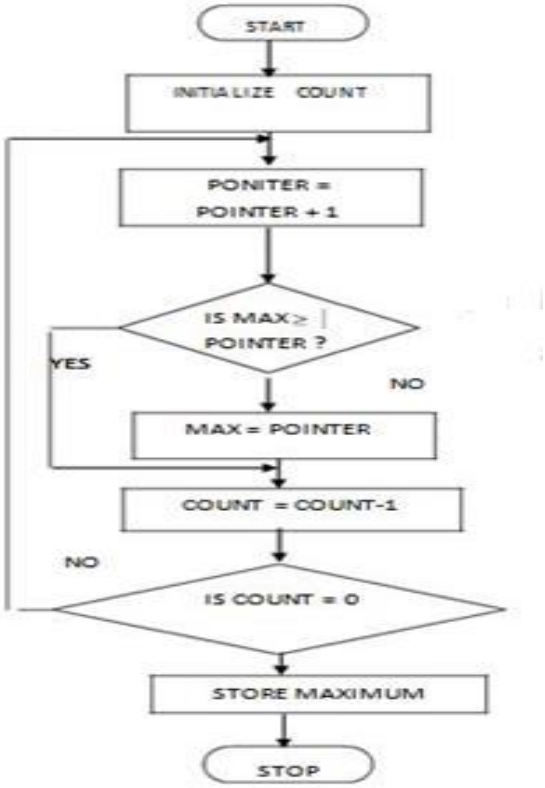
Step1: Load the array count in a register C.

Step2: Get the first two numbers

Step3: Compare the numbers and exchange if the number is large

Step4: Get the third number from the array and repeat the process until C is 0

FLOWCHART



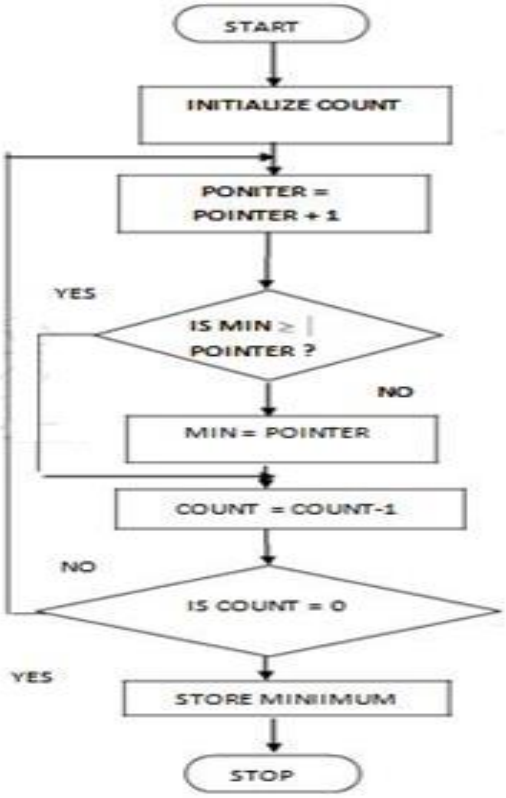


Fig: Finding the largest number

Fig: Finding the smallest number

IN LAB WORK
PROGRAM

Address	Label	Mnemonics	Comments
1000		MOV SI,2000H	Initialize the pointer
1003		MOV CL, [SI]	Initialize the count
1005		MOV CH,00H	Initialize value in CH register
1007		INC SI	Increment the address location
1008		MOV AL, [SI]	Move the SI pointer to AL register
100A		DEC CL	Reduce the iteration count
100C		INC SI	Increment the address location
100D	L2:	CMP AL,[SI]	Compare the array elements
100F		JNC/JC , L1 (1013)	For find largest number If AL > [SI] then go to L1 (no swap) For find smallest number If AL < [SI] then go to L1 (no swap)
1011		MOV AL,[SI]	Move the value in SI pointer to AL register
1013	L1:	INC SI	Increment the address location
1014		LOOP L2 (100D)	Go to location L2(100D)and repeat the loop
1016		MOV [2100],AL	Move the result in AL to a location and store.
101A		HLT	Stop the program

POST LAB WORK
Input:

Memory Location	Input Data
CL	03H
2000	0AH
2001	06H
2002	01H

Output (Largest Element)

Memory Location	Output Data
2100	0AH

Output (Smallest Element)

Memory Location	Output Data
2100	01H

Pre-Lab Viva Questions:

1. What is program counter?
2. What is the maximum clock frequency in 8086?
3. What are the various segments registers in 8086?
4. What are the different functional units in 8086?
5. What is the difference between JUMP and LOOP instructions?

Post-Lab Viva Questions:

1. What registers are typically used to store the largest or smallest number found in an array?
2. How would you optimize the code for finding the largest or smallest number in an array?
3. Write the process of direct memory address with one example.
4. Draw the flag register in 8086 microprocessors.
5. What are the addressing modes used in the above program?

RESULT:

Thus the program for developing a system to analyze student grades, by determining the highest and lowest grades in an array was executed and grading distribution was determined using 8086 trainer kit.

INTRODUCTION TO 8051 MICROCONTROLLER

Objective

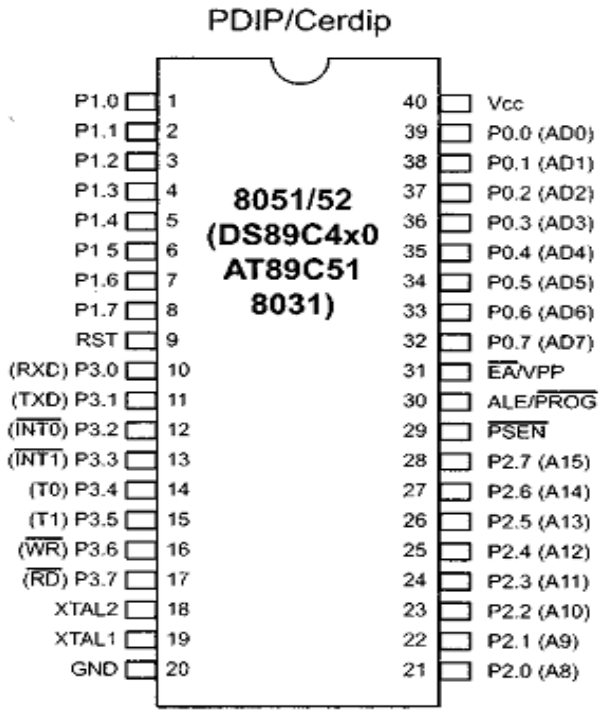
To study the components, functionality and overall operations of 8051 microcontroller architecture.

8051 MICROCONTROLLER

FEATURES:

- 8-bit ALU, Accumulator and 8-bit Registers; hence it is an 8-bit microcontroller
- 8-bit data bus - It can access 8 bits of data in one operation
- 16-bit address bus - It can access 2^{16} memory locations - 64 KB (65536 locations) each of RAM and ROM
- On-chip RAM - 128 bytes (data memory) and On-chip ROM - 4 kByte (program memory)
- Four byte bi-directional input/output port
- UART (serial port)
- Two 16-bit Counter/timers
- Two-level interrupt priority

PIN DIAGRAM OF 8051



PIN DESCRIPTION

Pins 1 – 8:- Recognized as Port 1. Different from other ports, this port doesn't provide any other purpose. Port 1 is a domestically pulled up, quasi bi directional Input/output port.

Pin 9:- As made clear previously RESET pin is utilized to set the micro-controller 8051 to its primary values, whereas the micro-controller is functioning or at the early beginning of application. The RESET pin has to be set elevated for two machine rotations.

Pins 10 – 17:- Recognized as Port 3. This port also supplies a number of other functions such as timer input, interrupts, serial communication indicators TxD & RxD, control indicators for outside memory interfacing WR & RD, etc. This is a domestic pull up port with quasi bi directional port within.

Pins 18 and 19:- These are employed for interfacing an outer crystal to give system clock.

Pin 20:- Titled as Vss – it symbolizes ground (0 V) association.

Pins- 21-28:- Recognized as Port 2 (P 2.0 – P 2.7) – other than serving as Input/output port, senior order address bus indicators are multiplexed with this quasi bi directional port.

Pin- 29:- Program Store Enable or PSEN is employed to interpret sign from outer program memory.

Pin-30:- External Access or EA input is employed to permit or prohibit outer memory interfacing. If there is no outer memory need, this pin is dragged high by linking it to Vcc.

Pin-31:- Address Latch Enable or ALE is brought into play to de-multiplex the address data indication of port 0 (for outer memory interfacing). Two ALE throbs are obtainable for every machine rotation.

Pins 32-39: Recognized as Port 0 (P0.0 to P0.7) – other than serving as Input/output port, low order data & address bus signals are multiplexed with this port (to provide the use of outer memory interfacing). This pin is a bi directional Input/output port (the single one in microcontroller 8051) and outer pull up resistors are necessary to utilize this port as Input/output.

Pin-40: Termed as Vcc is the chief power supply. By and large it is +5V DC.

8051 INSTRUCTION SET:
ARITHMETIC INSTRUCTIONS:

Mnemonic	Instruction	Description
ADD	A, #Data	A ← A + Data
	A, Rn	A ← A + Rn
	A, Direct	A ← A + (Direct)
	A, @Ri	A ← A + @Ri
ADDC	A, #Data	A ← A + Data + C
	A, Rn	A ← A + Rn + C
	A, Direct	A ← A + (Direct) + C
	A, @Ri	A ← A + @Ri + C
SUBB	A, #Data	A ← A – Data – C
	A, Rn	A ← A – Rn – C
	A, Direct	A ← A – (Direct) – C
	A, @Ri	A ← A – @Ri – C
MUL	AB	Multiply A with B (A ← Lower Byte of A*B and B ← Higher Byte of A*B)
DIV	AB	Divide A by B (A ← Quotient and B ← Remainder)

Logical Instructions:

Mnemonic	Instruction	Description
ANL	A, #Data	A ← A AND Data
	A, Rn	A ← A AND Rn
	A, Direct	A ← A AND (Direct)
	A, @Ri	A ← A AND @Ri
	Direct, A	(Direct) ← (Direct) AND A
	Direct, #Data	(Direct) ← (Direct) AND #Data
ORL	A, #Data	A ← A OR Data
	A, Rn	A ← A OR Rn
	A, Direct	A ← A OR (Direct)
	A, @Ri	A ← A OR @Ri
	Direct, A	(Direct) ← (Direct) OR A
	Direct, #Data	(Direct) ← (Direct) OR #Data
XRL	A, #Data	A ← A XRL Data
	A, Rn	A ← A XRL Rn
	A, Direct	A ← A XRL (Direct)
	A, @Ri	A ← A XRL @Ri
	Direct, A	(Direct) ← (Direct) XRL A
	Direct, #Data	(Direct) ← (Direct) XRL #Data

Data Transfer Instructions:

Mnemonic	Instruction	Description
MOV	A, #Data	A ← Data
	A, Rn	A ← Rn
	A, Direct	A ← (Direct)
	A, @Ri	A ← @Ri
	Rn, #Data	Rn ← data
	Rn, A	Rn ← A
	Rn, Direct	Rn ← (Direct)
	Direct, A	(Direct) ← A
	Direct, Rn	(Direct) ← Rn
	Direct1, Direct2	(Direct1) ← (Direct2)
	Direct, @Ri	(Direct) ← @Ri
	Direct, #Data	(Direct) ← #Data
	@Ri, A	@Ri ← A
	@Ri, Direct	@Ri ← Direct
	@Ri, #Data	@Ri ← #Data
	DPTR, #Data16	DPTR ← #Data16

MOVC	A, @A+DPTR	A ← Code Pointed by A+DPTR
	A, @A+PC	A ← Code Pointed by A+PC
	A, @Ri	A ← Code Pointed by Ri (8-bit Address)
MOVX	A, @DPTR	A ← External Data Pointed by DPTR
	@Ri, A	@Ri ← A (External Data 8-bit Addr)
	@DPTR, A	@DPTR ← A (External Data 16-bit Addr)
PUSH	Direct	Stack Pointer SP ← (Direct)
POP	Direct	(Direct) ← Stack Pointer SP
XCH	Rn	Exchange ACC with Rn
	Direct	Exchange ACC with Direct Byte
	@Ri	Exchange ACC with Indirect RAM
XCHD	A, @Ri	Exchange ACC with Lower Order Indirect RAM

Branch Instructions:

Mnemonic	Instruction	Description
ACALL	ADDR11	Absolute Subroutine Call PC + 2 → (SP); ADDR11 → PC
LCALL	ADDR16	Long Subroutine Call PC + 3 → (SP); ADDR16 → PC
RET	--	Return from Subroutine (SP) → PC
RETI	--	Return from Interrupt
AJMP	ADDR11	Absolute Jump ADDR11 → PC
LJMP	ADDR16	Long Jump ADDR16 → PC
SJMP	rel	Short Jump PC + 2 + rel → PC
JMP	@A + DPTR	A + DPTR → PC
JZ	rel	If A=0, Jump to PC + rel
JNZ	rel	If A ≠ 0, Jump to PC + rel

CJNE	A, Direct, rel	Compare (Direct) with A. Jump to PC + rel if not equal
	A, #Data, rel	Compare #Data with A. Jump to PC + rel if not equal
	Rn, #Data, rel	Compare #Data with Rn. Jump to PC + rel if not equal
	@Ri, #Data, rel	Compare #Data with @Ri. Jump to PC + rel if not equal
DJNZ	Rn, rel	Decrement Rn. Jump to PC + rel if not zero
	Direct, rel	Decrement (Direct). Jump to PC + rel if not zero
NOP		No Operation

8051 Microcontroller Hardware Kit:

Result:

Thus the components and functionality of 8051 Microcontroller was studied successfully.

Ex No: 2 Date:	BASIC ARITHMETIC OPERATIONS USING 8051 MICROCONTROLLERS

OBJECTIVE:

To write the assembly language program by implementing basic arithmetic operations to calculate speed, direction, and torque for motor control in robots and automated machinery using 8051 microprocessors.

SYSTEM AND SOFTWARE TOOL REQUIRED:

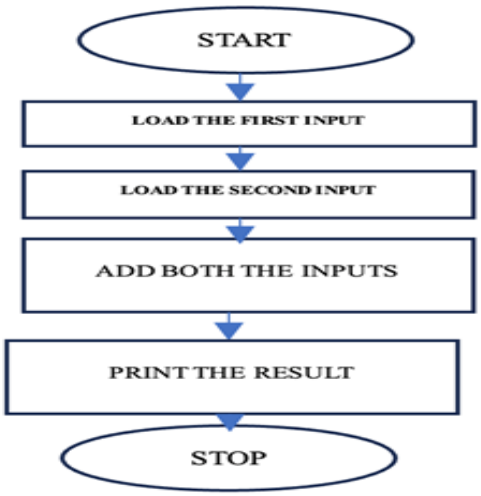
1. 8051 Microprocessor kit
2. Keyboard

PRE LAB-WORK
ADDITION OF TWO NUMBERS

ALGORITHM:

1. Start the program
2. Load the 1st number to A register.
3. Add 8-bit second data in A register.
4. Store the result in memory location.
5. Halt the program.

FLOWCHART



IN LAB WORK

PROGRAM: ADDITION

Address	Mnemonics	Comments
8500	MOV A, #13H	Move 8-bit first data to A register.
8502	ADD A, #14H	Add 8-bit second data in A register.
8504	MOV DPTR, #8600	Initialize memory pointer
8507	MOVX, @DPTR	Store the result
8508	HERE: SJMP HERE	End

POST LAB WORK
Input

Memory Location	Input Data
8500	13H
8502	14H

Output

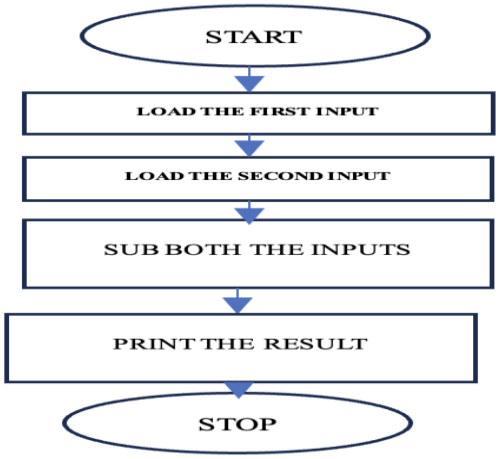
Memory Location	Output Data
8600	27H

SUBTRACTION OF TWO NUMBERS

ALGORITHM:

1. Start the program
2. Load the 1st number to A register.
3. Subtract 8-bit second data from A register.
4. Store the result in memory location
5. Halt the program.

FLOWCHART



IN LAB WORK
PROGRAM: SUBTRACTION

Address	Mnemonics	Comments
8500	MOV A, #20H	Move 8-bit first data to A register.
8502	SUBB A, #10H	Subtract 8-bit second data from A register.
8504	MOV DPTR, #8600	Initialize memory pointer
8507	MOVX, @DPTR	Store the result
8508	HERE:SJMP HERE	End

POST LAB WORK

Input

Memory Location	Input Data
8500	20H
8502	10H

Output

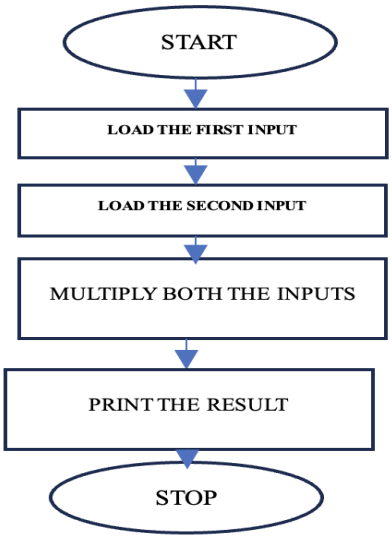
Memory Location	Output Data
8600	10H

MULTIPLICATION OF TWO NUMBERS

ALGORITHM:

1. Start the program
2. Load the 1st number to A register.
3. Load the 2nd number to B register.
4. Multiply both the numbers
5. Store the result in memory location
6. Halt the program.

FLOWCHART



IN LAB WORK

PROGRAM: MULTIPLICATION

Address	Mnemonics	Comments
8500	MOV A, #06H	Move 8-bit first data to A register.
8502	MOV F0, #03H	Move 8-bit second data to B register.
8505	MUL AB	Multiply the data.
8506	MOV DPTR, #8600	Initialize memory pointer
8509	MOVX, @DPTR, A	Store the result
850A	INC DPTR	Increment memory pointer
850B	MOV A,F0	Get the packed BCD number
850D	MOVX, @DPTR, A	Store the result
850E	HERE: SJMP HERE	End

POST LAB WORK
Input

Memory Location	Input Data
8500	06H
8502	03H

Output

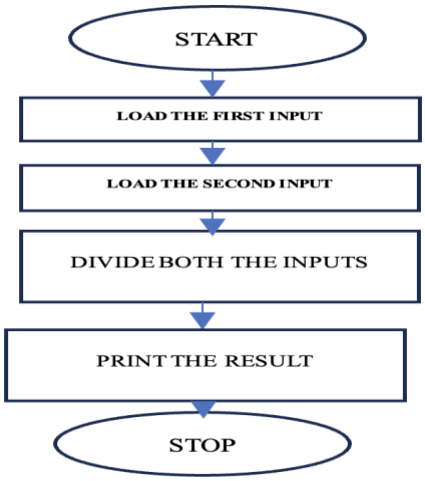
Memory Location	Output Data
8600	12H

DIVISION OF TWO NUMBERS

ALGORITHM:

1. Start the program
2. Load the 1st number to AX register.
3. Load the 2nd number to BX register.
4. Divide both the numbers
5. Store the result in memory location
6. Halt the program.

FLOWCHART



IN LAB WORK

PROGRAM: DIVISION

Address	Mnemonics	Comments
8500	MOV A, #09H	Move 8-bit first data to A register.
8502	MOV F0, #03H	Move 8-bit second data to B register.
8505	DIV AB	Divide the data.
8506	MOV DPTR, #8600	Initialize memory pointer
8509	MOVX, @DPTR, A	Store the result
850A	INC DPTR	Increment memory pointer
850B	MOV A,F0	Get the packed BCD number
850D	MOVX, @DPTR, A	Store the result
850E	HERE: SJMP HERE	End

POST LAB WORK

Input

Memory Location	Input Data
8500	09H
8502	03H

Output

Memory Location	Output Data
8600	03H

Application:

1. Robotic Arm Control.
2. Calculator Application.
3. Filtering, FFT and other signal processing tasks.
4. Audio processing applications.
5. Analyze weather data.

Pre-Lab Viva Questions:

1. Define microcontroller and mention few applications.
2. Compare microprocessor and microcontroller.
3. List the instruction set used in 8051 micro controllers.
4. With an example explain direct addressing mode and indirect addressing mode in 8051 microcontrollers.
5. Write the function of #DPTR.

Post-Lab Viva questions:

1. MOVR0,#20H, MOVA,@R0 and MOVR5,A- Differentiate the functions of the three instructions works?
2. Take two numbers, 5FH and D8H, at locations 20H and 21H; after adding them, the result will be stored at locations 30H and 31H: Use the same program, replace it with the data given in the question and explain the program step by step.
3. Manually calculate the values and compare with the results obtained from the microcontroller.
4. What would you do if a division by zero occurs in an 8051 program?
5. How does the 8051 handle overflow in multiplication and division?

RESULT:

Thus, the assembly language program for implementing basic arithmetic operations to calculate speed, direction, and torque for motor control in robots and automated machinery is performed using 8051 microprocessors.

Ex No: 3 Date:	STEPPER MOTOR CONTROL USING 8086 MICROPROCESSOR

OBJECTIVE:

To write an assembly language program using stepper motor for controlling the speed and position of the conveyor belt in manufacturing or packaging industries for efficient item transport and sorting using 8086 emulator.

SYSTEM AND SOFTWARE TOOL REQUIRED:

1. 8086 emulator (EMU8086)

PROCEDURE:

1. Open the Emulator window and create a new .asm file
2. Type the program for stepper motor control
3. Click on the emulator icon, a new dialog box will open.
4. Click the run icon, program will now be executed.
5. Simulation window of Stepper motor will appear now and rotation of stepper motor can be observed there.

PROGRAM

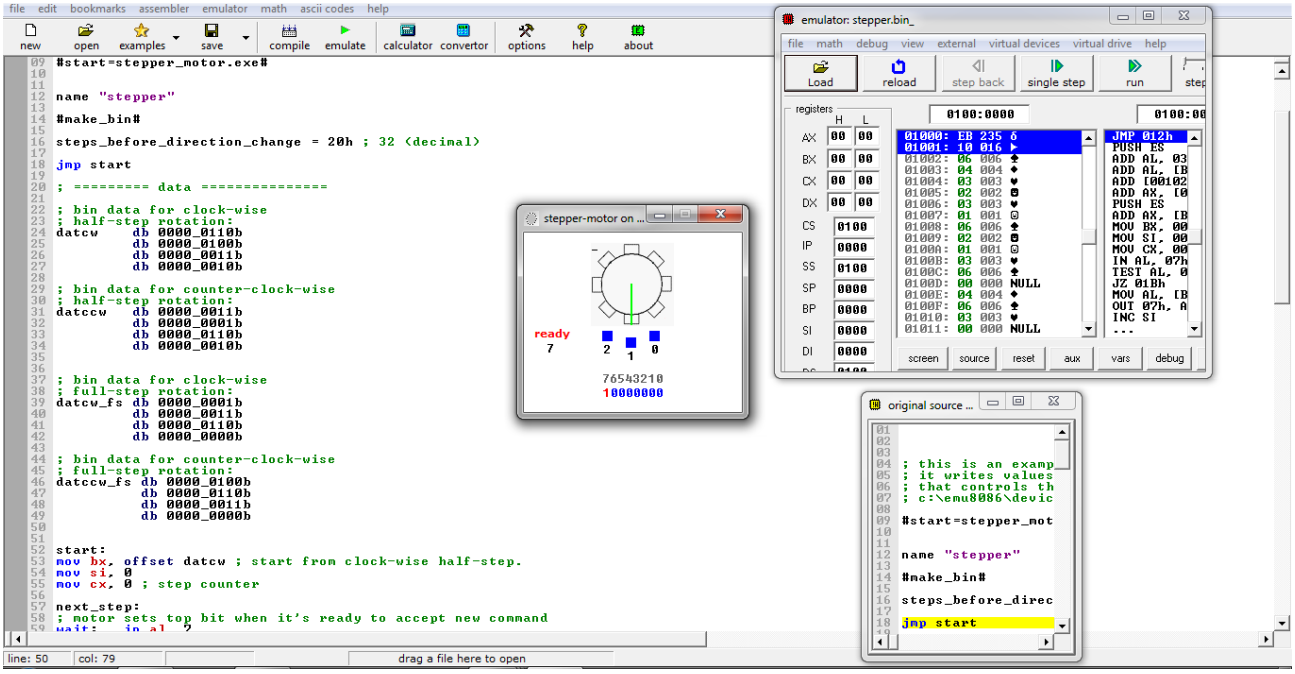
```
#start=stepper_motor.exe#
name "stepper"
#make_bin#

steps_before_direction_change = 20h ; 32 (decimal)
jmp start
; ========= data ===============
; bin data for clock-wise
; half-step rotation:
datcw    db 0000_0110b
         db 0000_0100b
         db 0000_0011b
         db 0000_0010b

; bin data for counter-clock-wise
; half-step rotation:
datccw   db 0000_0011b
         db 0000_0001b
         db 0000_0110b
         db 0000_0010b

; bin data for clock-wise
; full-step rotation:
datcw_fs  db 0000_0001b
          db 0000_0011b
          db 0000_0110b
          db 0000_0000b
```

```
; bin data for counter-clock-wise
; full-step rotation:
datccw_fs db 0000_0100b
         db 0000_0110b
         db 0000_0011b
         db 0000_0000b

start:
mov bx, offset datcw ; start from clock-wise half-step.
mov si, 0
mov cx, 0 ; step counter

next_step:
; motor sets top bit when it's ready to accept new command
wait:   in al, 7
      test al, 10000000b
      jz wait

mov al, [bx][si]
out 7, al
inc si
cmp si, 4
jb next_step
mov si, 0
inc cx

cmp cx, steps_before_direction_change
jb  next_step
mov cx, 0
add bx, 4 ; next bin data
cmp bx, offset datccw_fs
jbe next_step
mov bx, offset datcw ; return to clock-wise half-step.
jmp next_step
```

Application:

1. Automated drug delivery systems.
2. Automated Conveyor Systems.
3. Telescope Positioning System.
4. Automated Stage Lighting.
5. Home Automation Systems.

Simulation Results



Pre-Lab Viva Questions:

1. Define microprocessor.
2. Write any 4 addressing modes used in 8086 microprocessors.
3. List the applications of microprocessor.
4. Define the term BIU and EU.
5. Mention any four types of instruction set used in 8086 microprocessors.

Post-Lab Viva questions:

1. Why do we need to use a delay in the stepper motor control program?
2. How does the OUT instruction work in the context of controlling the stepper motor?
3. What is the purpose of the ORG directive in the assembly program?
4. Why do we need to initialize the DX register with the port address?
5. What is the function of the CALL DELAY instruction in the program?

RESULT:

Thus the program using stepper motor for controlling the speed and position of the conveyor belt in manufacturing or packaging industries was executed using 8086 emulator.

Ex No: 4 Date:	SENSOR INTERFACING USING 8051 MICROCONTROLLER.

OBJECTIVE:

To write an assembly language program for developing security systems for intrusion detection and monitoring using motion detectors, door/window contact sensors, and smoke detector sensor with 8051 microcontroller using Edsim 51 software.

SYSTEM AND SOFTWARE TOOL REQUIRED:

1. Edsim 51 simulation software

PROCEDURE

1. Open the edsim51 software.
2. Type the program in the new window
3. Click Assemble, simulator will check for errors and assigns address for each byte of instruction
4. View the result in registers and graph in oscilloscope.
5. For various values of inputs check the results

PROGRAM

i) Digital to Analog Converter

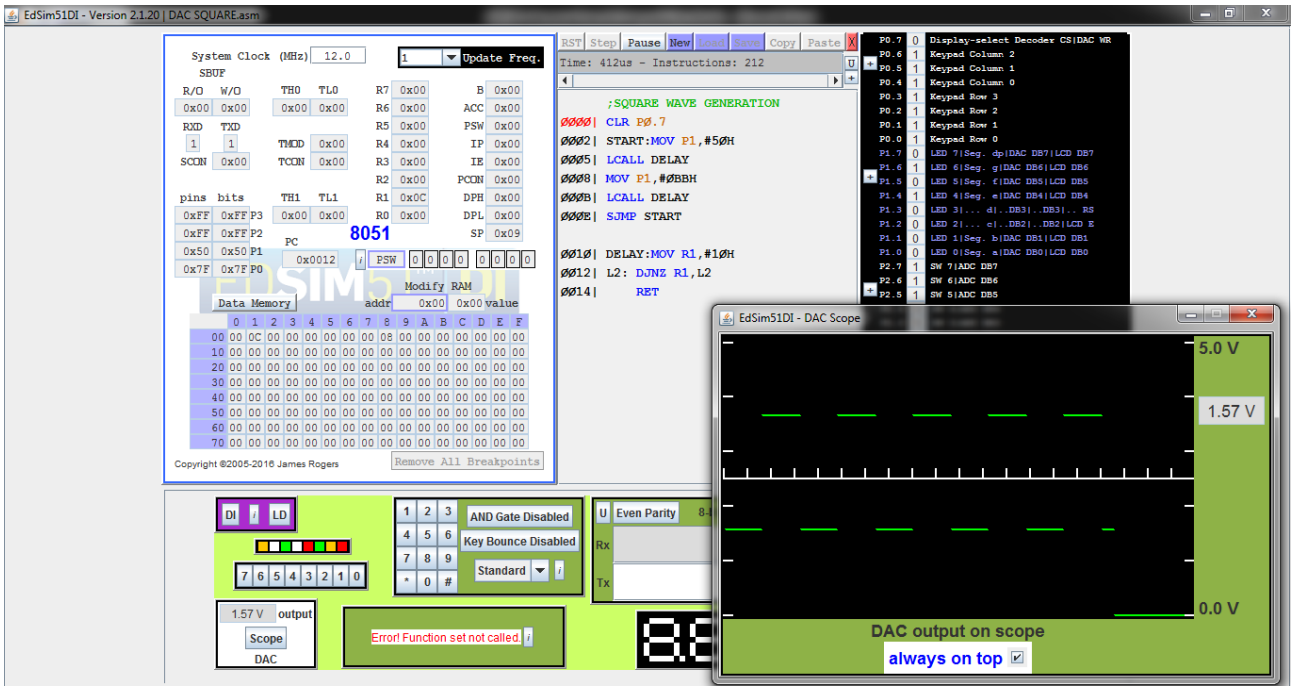
Memory	Label	Mnemonics	Comment
0000		CLR P0.7	
0002	START	MOV P1, #50H	Load the initial value for T_{off} of a square wave
0005		LCALL DELAY	Call the delay
0008		MOV P1, #0BBH	Load the Value for T_{on}
000B		LCALL DELAY	Call the delay
000E		SJMP START	Repeat
0010	DELAY	MOV R1, #45H	Move the delay value to R1
0012	L2	DJNZ R1, L2	Decrement and repeat
0014		RET	Return to main program

ii) Echoing the Switches on the LEDs

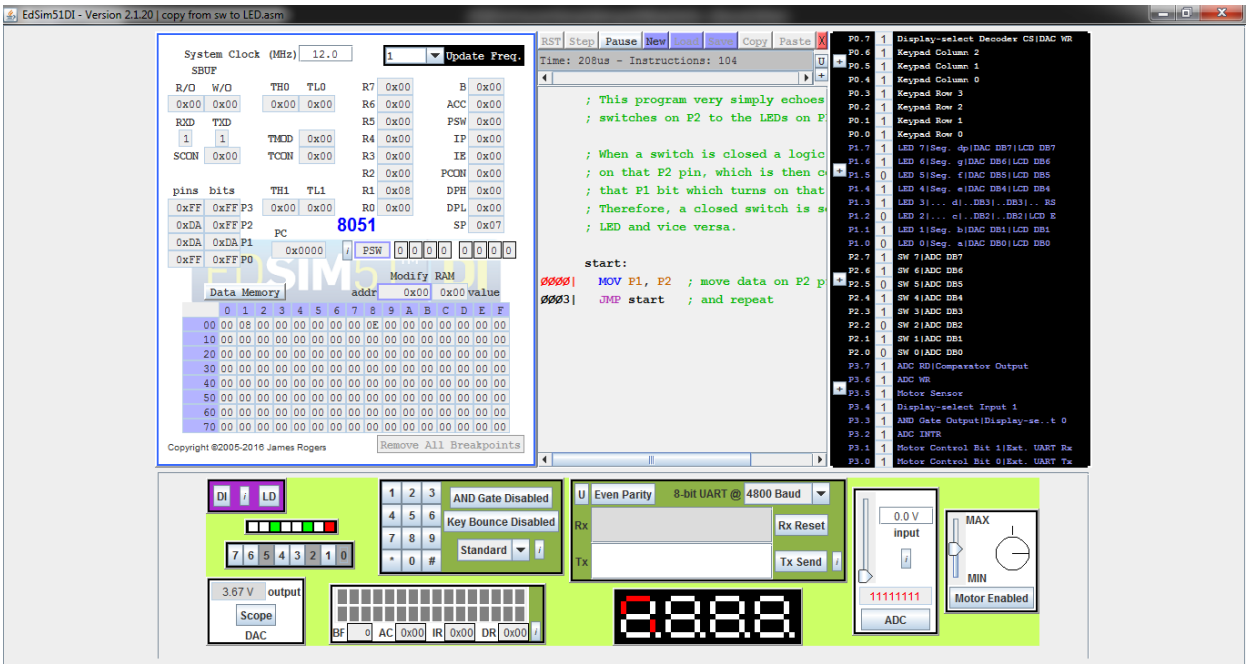
Memory	**Label**	**Mnemonics**	**Comment**
0000	START	MOV P1, P2	Move data on Port 2 pins to Port1
0003		JMP START	Repeat

Simulation Results

i) Digital to Analog Converter



ii) Echoing the Switches on the LEDs



Application:

1. Monitor and report environmental conditions.
2. Automate household functions.
3. Industrial Automation System.
4. Health Monitoring System.
5. Agricultural Automation System.

Pre-Lab Viva Questions:

1. Write the types of DAC.
2. Why is an oscilloscope used to verify the DAC output?
3. What kind of waveform did you observe on the oscilloscope when testing the DAC output?
4. Sketch the block diagram of D/A converter.
5. Interpret resolution and accuracy of a D/A converter.

Post-Lab Viva Questions:

1. How is the DAC interfaced with the 8051 microcontroller?
2. What is the purpose of using a DAC in microcontroller applications?
3. The basic step of a 9 bit DAC is 10.3 mV. If 000000000 represents 0Volts, what is the output for an input of 101101111?
4. How did you sense the inputs from the switches and reflect them on the LEDs?
5. Describe the purpose of the MOV and OUT instructions in your assembly program for the 8051 microcontroller.

RESULT:

Thus an assembly language program for developing security systems for intrusion detection and monitoring system using DAC interfaced with 8051 microcontroller was verified in oscilloscope and inputs from switches were sensed and reflected in LEDs.

Ex No:5 Date:	INTERFACING 8051 WITH ADC

OBJECTIVE:

To write an assembly language program for developing irrigation systems by interfacing soil moisture sensors with an ADC to convert analog moisture levels into digital signals with 8051 microcontroller using Edsim 51 software.

SYSTEM AND SOFTWARE TOOL REQUIRED:

1. Edsim 51 simulation software

PROCEDURE

1. Open the edsim51 software.
2. Type the program in the new window
3. Click Assemble, simulator will check for errors and assigns address for each byte of instruction
4. Adjust the Potentiometer and note down the corresponding digital output in ADC section of software.
5. For various values of inputs in potentiometer check the results

PROGRAM

Analog to Digital Converter

```
ORG 0                   ; reset vector
        JMP main        ; jump to the main program

ORG 3                   ; external 0 interrupt vector
        JMP ext0ISR     ; jump to the external 0 ISR

ORG 0BH                 ; timer 0 interrupt vector
        JMP timer0ISR   ; jump to timer 0 ISR

ORG 30H                 ; main program starts here
main:
        SETB IT0        ; set external 0 interrupt as edge-activated
        SETB EX0        ; enable external 0 interrupt
        CLR P0.7        ; enable DAC WR line
        MOV TMOD, #2    ; set timer 0 as 8-bit auto-reload interval timer

        MOV TH0, #-50   ; | put -50 into timer 0 high-byte

        MOV TL0, #-50   ; | put the same value in the low byte to ensure the timer starts

        SETB TR0        ; start timer 0
        SETB ET0        ; enable timer 0 interrupt
        SETB EA         ; set the global interrupt enable bit
        JMP $           ; jump back to the same line
```

; end of main program

; timer 0 ISR - simply starts an ADC conversion
timer0ISR:

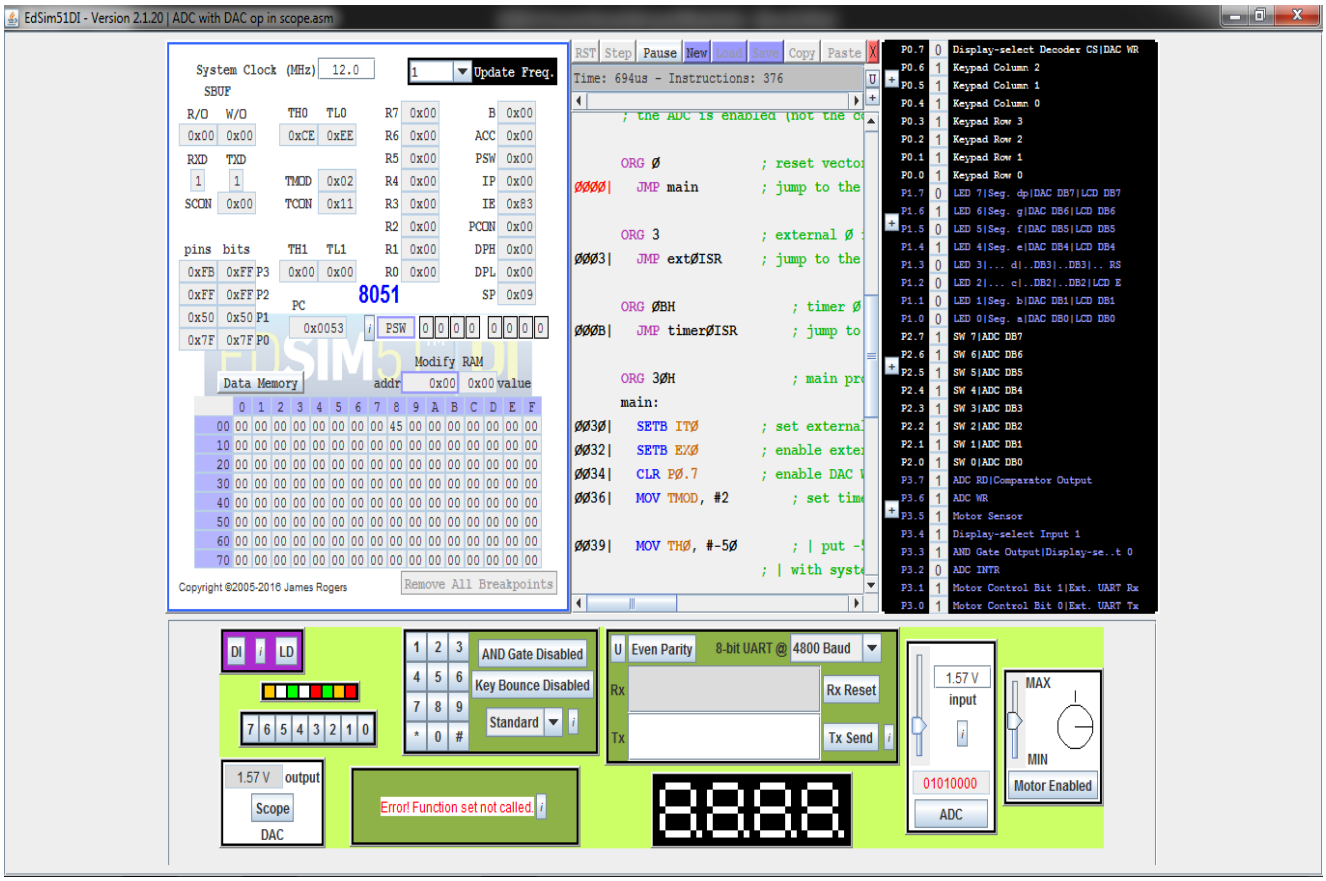
```
CLR P3.6        ; clear ADC WR line
SETB P3.6       ; then set it - this results in the required positive edge to start a
                  conversion
RETI            ; return from interrupt
```

; external 0 ISR - responds to the ADC conversion complete interrupt

ext0ISR:

```
CLR P3.7        ; clear the ADC RD line - this enables the data lines
MOV P1, P2      ; take the data from the ADC on P2 and send it to the DAC data lines
                  on P1
SETB P3.7       ; disable the ADC data lines by setting RD
RETI            ; return from interrupt
```

Simulation Results



Application:

1. Data Acquisition System.
2. Energy Monitoring System.
3. Automotive Applications.
4. Battery Monitoring System.
5. Medical Monitoring System.

Pre-Lab Viva Questions:

1. List the common types of ADC.
2. Mention the applications of ADC.
3. How ADC is classified?
4. What is the application of ADC in microphone?
5. Define start of conversion and end of conversion.

Post-Lab Viva Questions:

1. What is the purpose of using an ADC in microcontroller applications?
2. How do you ensure that the analog input signal is stable before starting the conversion process?
3. What types of ADCs are commonly used, and which type was used in your experiment?
4. Define the role of sampling rate in ADC operation.
5. What modifications would be needed in the program to interface a different type of ADC with the 8051 microcontroller?

RESULT:

Thus an assembly language program for developing irrigation systems was implemented using ADC interfaced with 8051 microcontroller using Edsim 51 software.

Ex No:6 Date:	BASIC PROGRAM WITH ARDUINO KIT

OBJECTIVE:

To write a program to blink an LED on and off at regular intervals and control the brightness of an LED based on ambient light levels using Arduino.

SYSTEM AND SOFTWARE TOOL REQUIRED:

1. Arduino board
2. Arduino IDE

PROGRAM 1 - LED BLINK

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);     // turn the LED on (HIGH is the voltage level)
  delay(1000);                         // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                        // wait for a second
}
```

EXPLANATION

1. We start by declaring a constant variable LED_BUILTIN and assigning it the value 13, which corresponds to the pin number of the built-in LED on the Arduino UNO.

2. In the setup() function, we set the LED_BUILTIN as an output using the pinMode() function. This configuration is necessary to indicate that we want to control the LED connected to that pin.

3. The loop() function is where the main logic of the program resides. Inside the loop(), we toggle the LED on and off with a delay in between.

4. First, we turn the LED on by setting the LED_BUILTIN to HIGH using the digitalWrite() function.

5. We then wait for 1 second using the delay() function to keep the LED on.

6. Next, we turn the LED off by setting the LED_BUILTIN to LOW using the digitalWrite() function.

7. We again wait for 1 second using the delay() function to keep the LED off.

8. The program then goes back to the beginning of the loop() function and repeats the process indefinitely, resulting in the LED blinking on and off at a 1-second interval.

9. By uploading this program to an Arduino UNO, you will see the built-in LED on pin 13 blinking on and off repeatedly.

PROGRAM 2 - LED FADE

```
const int ledPin = 9;

const int fadeRate = 5;

void setup() {
  pinMode(ledPin, OUTPUT);
}

void loop() {
  for (int brightness = 0; brightness <= 255; brightness++)
{
    analogWrite(ledPin, brightness);
    delay(fadeRate);
  }

  for (int brightness = 255; brightness >= 0; brightness--)
{
    analogWrite(ledPin, brightness);
    delay(fadeRate);
  }
}
```

EXPLANATION

1. We start by declaring a constant variable ledPin and assigning it the value 9, which corresponds to the pin number where the LED is connected. This pin must support Pulse Width Modulation (PWM) for fading the LED.

2. We also declare a constant variable fadeRate and set it to 5. This value represents the delay in milliseconds between each step of the fade.

3. In the setup() function, we set the ledPin as an output using the pinMode() function, just like in the previous example.

4. The main logic of the program resides in the loop() function. Inside the loop(), we have two for loops that gradually increase and decrease the brightness of the LED, creating a fade effect.

5. In the first for loop, we start with a brightness value of 0 and gradually increase it up to 255. We use the analogWrite() function to set the brightness of the LED connected to ledPin. The analogWrite() function takes a value between 0 and 255, where 0 is the minimum brightness

(off) and 255 is the maximum brightness (full on). We then introduce a delay of fadeRate milliseconds using the delay() function to control the speed of the fade.

6. After the first for loop completes, the LED will be at its maximum brightness (full on).

7. In the second for loop, we start with a brightness value of 255 and gradually decrease it down to 0. Similar to the previous loop, we use analogWrite() to set the brightness and introduce a delay of fadeRate milliseconds.

8. After the second for loop completes, the LED will be at its minimum brightness (off).

9. The program then goes back to the beginning of the loop() function and repeats the fade effect indefinitely, continuously fading the LED up and down.

10. By uploading this program to an Arduino UNO, you will see the LED connected to pin 9 gradually fading in and out, creating a smooth and continuous transition of brightness

Application:

1. Button Controlled LED.
2. Potentiometer Controlled LED Brightness.
3. Temperature-Based Fan Control.
4. Simple Light Show.
5. Light Sensitive LED

Pre-Lab Viva Questions:

1. What Arduino means?
2. Why Arduino is called microcontroller?
3. Compare and contrast Arduino and Raspberry pi
4. List the features of Arduino UNO?
5. Mention the main types of Arduino.

Post-Lab Viva Questions:

1. List the difference between digital and analog control of LEDs.
2. How is the inbuilt LED on pin 13 controlled in the program?
3. What is the purpose of using the delay function in the blinking LED program?
4. What is the role of the analogWrite function in creating the fading effect on pin 9?
5. How can you synchronize multiple LEDs to create complex light patterns?

RESULT:

Thus,

a) The inbuilt LED in pin 13 was made to blink ON and OFF repeatedly.
b) The LED controlled through pin 9 was made to fade in and out, creating a smooth and continuous transition of brightness based on ambient temperature.

Ex No: 7 Date:	DESIGN OF A TRAFFIC LIGHT CONTROLLER USING ARDUINO

OBJECTIVE:

To design and implement a traffic light control system using an Arduino microcontroller that manage the timing and sequencing of traffic lights for different directions, including handling pedestrian crossings and traffic flow optimization.

SYSTEM AND SOFTWARE TOOL REQUIRED:

1. Arduino board
2. Simulation software

A traffic light controller using Arduino is a popular project that demonstrates the use of Arduino microcontrollers for controlling traffic lights.

Explanation

1. Hardware Setup:
   - Arduino Board: Choose an appropriate Arduino board such as Arduino Uno or Arduino Mega.
   - Traffic Lights: Connect three different colour LEDs (red, yellow, green) to digital output pins of the Arduino board.
   - Resistors: Connect current-limiting resistors in series with each LED to prevent excessive current flow.

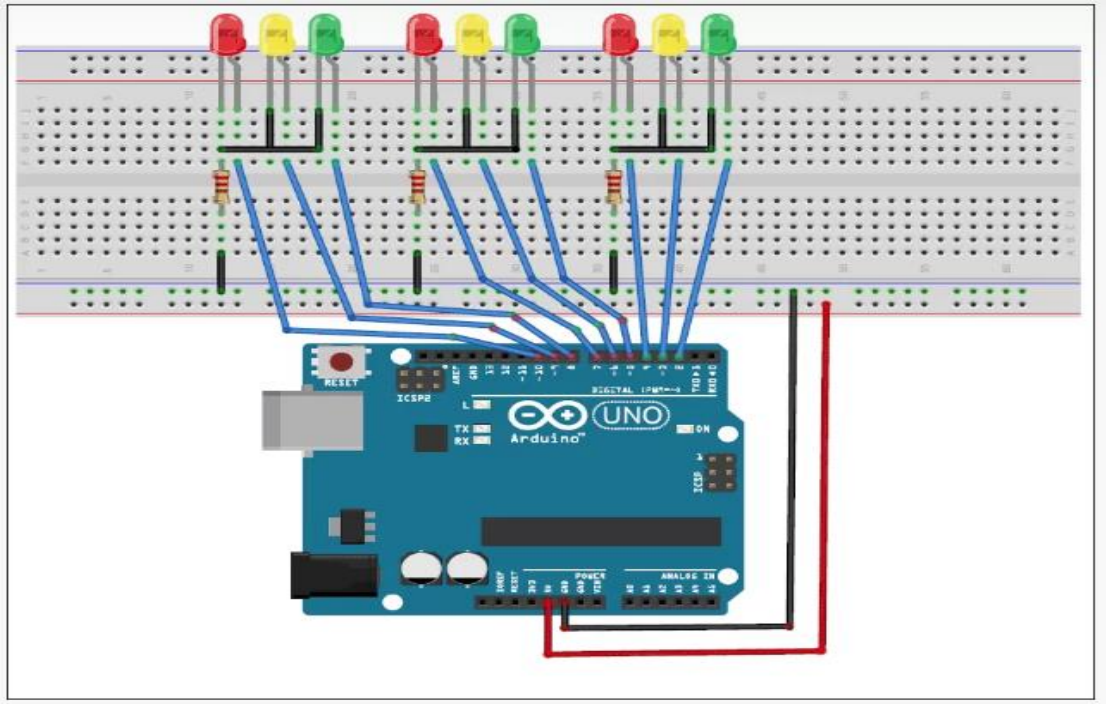
2. Programming Logic:
   - Open the Arduino Integrated Development Environment (IDE) and create a new sketch.
   - Define the digital pins connected to each LED as output pins using the `pinMode()` function.
   - Implement a traffic light control logic using a combination of `digitalWrite()` to turn on/off LEDs and `delay()` to control the timing.
   - Typically, the traffic light sequence follows a pattern of red > red + yellow > green > yellow > red.

3. Traffic Light Sequence:
   - Start with all lights off.
   - Turn on the red light and keep it on for a certain duration (e.g., 10 seconds) using `digitalWrite()` to set the corresponding pin HIGH.
   - Turn off the red light and turn on both the red and yellow lights together for a brief duration (e.g., 2 seconds) using `digitalWrite()`.
   - Turn off the red and yellow lights and turn on the green light for a specific duration (e.g., 15 seconds) using `digitalWrite()`.
   - Turn off the green light and turn on the yellow light for a short duration (e.g., 2 seconds) using `digitalWrite()`.
   - Repeat the sequence by going back to the red light.

4. Add Pedestrian Crossing:
   - Extend the traffic light controller to include a pedestrian crossing signal.
   - Connect additional LEDs (red and green) to represent pedestrian signals and define the corresponding pins as output.
   - Incorporate a pedestrian crossing logic, such as allowing pedestrians to cross when the green light is off and the red pedestrian light is on.

5. Compile and Upload:
   - Compile the Arduino sketch to check for any errors.
   - Upload the compiled code to the Arduino board using a USB cable.
6. Test and Refine:
   - Power on the Arduino and observe the traffic light sequence and pedestrian signals.
   - Make adjustments to the timing or add additional features based on the specific requirements of the traffic light controller project.

Circuit Diagram:

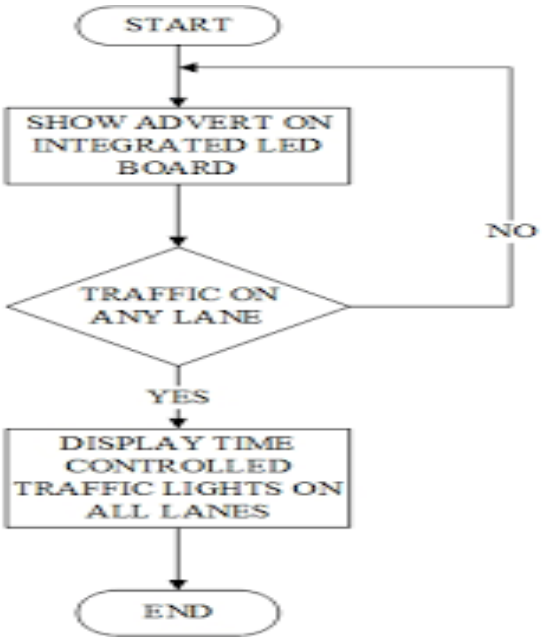


Code:

```
void setup() {
pinMode(A0,OUTPUT);
pinMode(A1,OUTPUT);
pinMode(A2,OUTPUT);
Serial.begin(9600);
}

void loop() {
digitalWrite(A0,HIGH);
digitalWrite(A1,LOW);
digitalWrite(A2,LOW);
Serial.println("STOP");
delay(5000);

digitalWrite(A0,LOW);
digitalWrite(A1,HIGH);
digitalWrite(A2,LOW);
Serial.println("Ready");
delay(2000);

digitalWrite(A0,LOW);
digitalWrite(A1,LOW);
digitalWrite(A2,HIGH);
```

```
Serial.println("Go");
delay(5000);
}
```

FLOCHART:



Pre-Lab Viva Questions:

1. What is the aim of traffic light controller?
2. List the major components of computerised traffic control system.
3. Which sensor is used in traffic light controller?
4. How are traffic lights activated?

Post-Lab Viva Questions:

1. What are the main components used in your traffic light controller system?
2. How can you enhance the traffic light controller to handle emergency vehicle detection or priority signalling?
3. What potential improvements could be made to your traffic light controller to increase its functionality or efficiency?
4. List the role of any external sensors or inputs used in the traffic light simulation.
5. How would you modify your design to include pedestrian signals and crosswalks?

RESULT:

Thus a traffic light control system using an Arduino microcontroller that manage the timing and sequencing of traffic lights for different directions was implemented and the sequence of traffic patterns was monitored.

Ex No: 8 Date:	DESIGN A SIMPLE CHAT SERVER USING ARDUINO

OBJECTIVE:

To design and implement a basic chat server system using Arduino microcontrollers to establish a simple communication protocol over a serial interface, allowing for real-time text messaging between multiple Arduino-based chat clients.

SYSTEM AND SOFTWARE TOOL REQUIRED:

1. Arduino board
2. Simulation software

Designing a chat server using Arduino involves creating a system that allows multiple clients to connect to the Arduino board and exchange messages.

Explanation:

1. Hardware Setup:

- Arduino Board: Choose an Arduino board with built-in Ethernet capabilities (e.g., Arduino Ethernet Shield or Arduino Ethernet Rev3).
- Ethernet Connection: Connect the Arduino board to the local network using an Ethernet cable.
- Power Supply: Provide power to the Arduino board using an appropriate power source.

2. Software Development:

- Arduino Sketch: Write the Arduino sketch to handle the server-side functionality.
- Ethernet Library: Utilize the Arduino Ethernet library to manage network communication.
- Server Setup: Configure the Arduino board as a server using the `Server` class from the Ethernet library.
- Client Connections: Accept incoming client connections using the `Server` object and establish communication channels.
- Message Handling: Implement code to receive and send messages between the server and clients using the appropriate methods (e.g., `client.available()`, `client.read()`, `client.write()`).
- Message Queuing: Implement a data structure to queue incoming messages from clients and process them sequentially.
- Client Tracking: Maintain a list of connected clients, their IP addresses, and other relevant information.

3. Communication Protocol:

- Define a simple protocol for client-server communication. For example:
- Clients send messages in a specific format (e.g., JSON) containing the sender, recipient, and message content.

- The server parses incoming messages, performs any required operations (e.g., message forwarding), and sends appropriate responses.
- Clients can request a list of connected clients or specific client information from the server.

4. Error Handling:

- Implement error handling mechanisms to handle issues such as client disconnections, message parsing errors, and network interruptions.
- Gracefully handle these errors to ensure the stability and reliability of the chat server.

5. Security Considerations:

- Depending on the requirements, consider implementing security measures such as authentication and encryption to protect the communication between the server and clients.

6. Testing and Deployment:

- Compile the Arduino sketch to check for any errors.
- Upload the sketch to the Arduino board.
- Connect the Arduino board to power and the local network.
- Test the chat server by connecting multiple clients and exchanging messages.
- Deploy the Arduino board in the desired environment, ensuring appropriate power and network connectivity.

Code:

```
/*
 Chat Server:  A simple server that distributes any incoming messages to all connected clients.  To use telnet to your device's IP address and type. You can see the client's input in the serial monitor as well.
 Using an Arduino Wiznet Ethernet shield.

 Circuit:
 * Ethernet shield attached to pins 10, 11, 12, 13
 * Analog inputs attached to pins A0 through A5 (optional)

#include <SPI.h>
#include <Ethernet.h>

// Enter a MAC address and IP address for your controller below.
// The IP address will be dependent on your local network.
// gateway and subnet are optional:
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress ip(192,168,1, 177);
IPAddress gateway(192,168,1, 1);
IPAddress subnet(255, 255, 0, 0);
```

```
// telnet defaults to port 23
EthernetServer server(23);
boolean gotAMessage = false; // whether or not you got a message from the client yet

void setup() {
  // initialize the ethernet device
  Ethernet.begin(mac, ip, gateway, subnet);
  // start listening for clients
  server.begin();
  // open the serial port
  Serial.begin(9600);
}

void loop() {
  // wait for a new client:
  EthernetClient client = server.available();

  // when the client sends the first byte, say hello:
  if (client) {
    if (!gotAMessage) {
      Serial.println("We have a new client");
      client.println("Hello, client!");
      gotAMessage = true;
    }

    // read the bytes incoming from the client:
    char thisChar = client.read();
    // echo the bytes back to the client:
    server.write(thisChar);
    // echo the bytes to the server as well:
    Serial.print(thisChar);
  }
}
```

RESULT:

Thus a basic chat server system was implemented using Arduino microcontrollers and also illustrates the serial communication between client-server interactions using Arduino hardware.

Ex No:9 Date:	BASIC PROGRAMMING USING ARM PROCESSOR USING KEIL C

OBJECTIVE:

To design and implement an embedded C program for interfacing an LED with an ARM microcontroller, and to verify the functionality of the LED by demonstrating various light patterns.

SYSTEM AND SOFTWARE TOOL REQUIRED:

1. ARM Evaluation Kit
2. ARM Development tools
3. LED
4. KEIL C software

THEORY

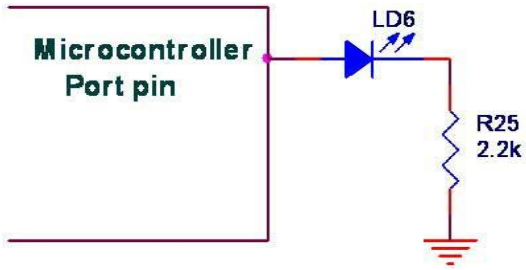
NXP's ARM7 (LPC2148), ARM Primer Kit is proposed to smooth the progress of developing and debugging of various designs encompassing of High speed 32-bit Microcontrollers.

LED (LIGHT EMITTING DIODES)

Light Emitting Diodes (LED) is the most commonly used components, usually for displaying pins digital states. Typical uses of LEDs include alarm devices, timers and confirmation of user input such as a mouse click or keystroke.

INTERFACING LED

Figure shows how to interface the LED to microcontroller. As you can see the Anode is connected through a resistor to GND & the Cathode is connected to the Microcontroller pin. So when the Port Pin is HIGH the LED is OFF & when the Port Pin is LOW the LED is turned ON.



ALGORITHMS

Step 1: Start the program
Step 2: Enable Input and Output Ports Step 3: Initialize timer
Step 4: Place the data for the LED in the data bus
Step 5: Call delay
Step 6: Rotate the data for the LEDs to switch on next LED
Step 7: Use infinite loops for the continuous blinking of LEDs
Step 8: Stop the program

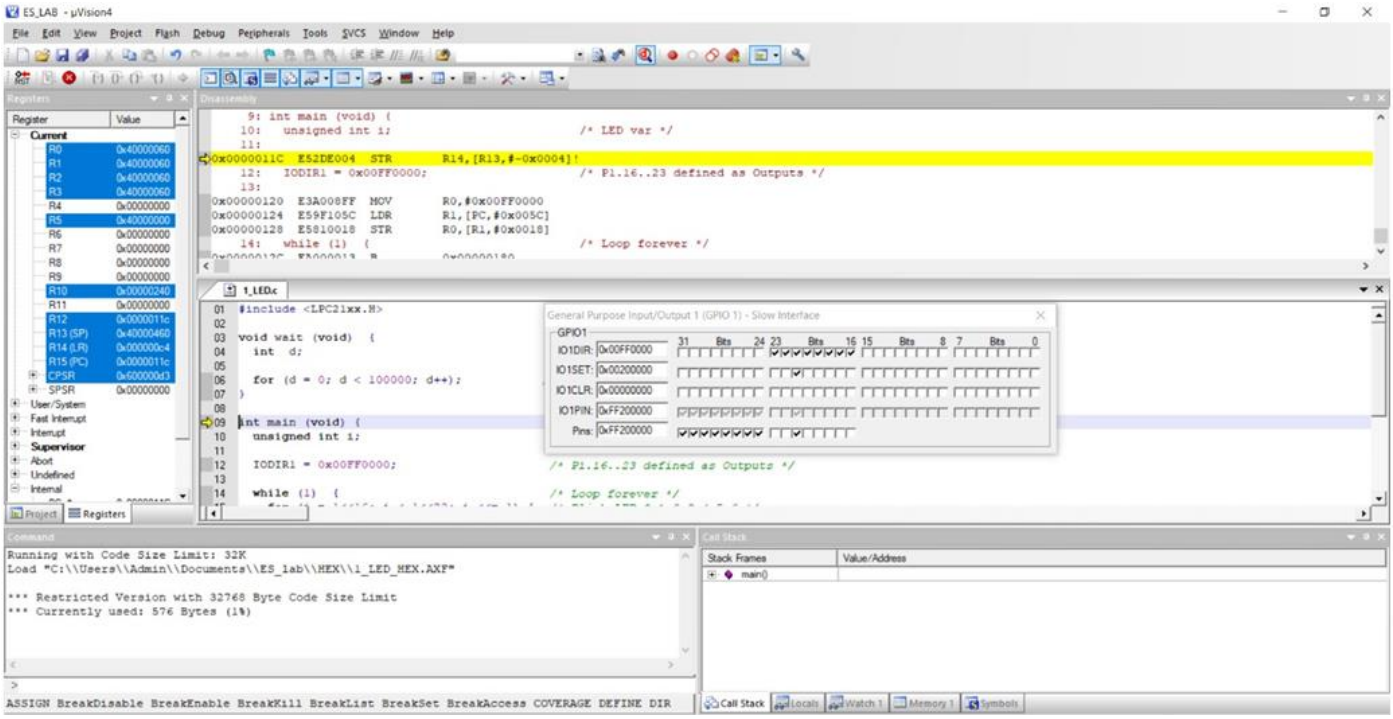
PROGRAM: 1 LED Blink

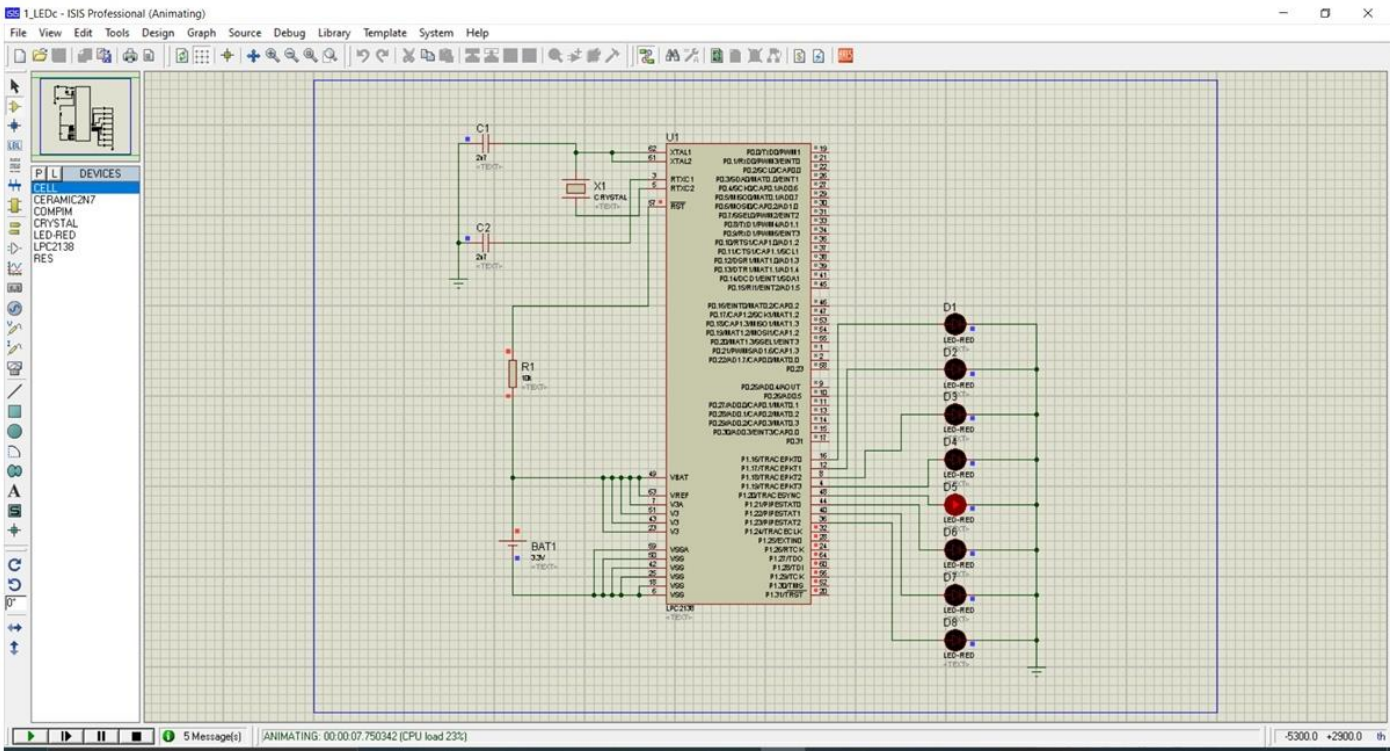
```
#include <LPC214x.H>
unsigned int delay;
int main(void)
{
IO0DIR = (1<<20);                                   // Configure P1.20 as Output
while(1)
{
IO0CLR = (1<<20);                                   // CLEAR (0) P1.20 to turn LED ON
for(delay=0; delay<500000; delay++);                // delay
IO0SET = (1<<20); // SET (1) P1.10 to turn LEDs OFF
for(delay=0; delay<500000; delay++);                // delay
}
}
```

PROGRAM: 2 LED Switch

```
#include <lpc214x.h>
int main(void)
{
    IO1DIR &= ~(1<<16);                  // explicitly making P1.16 as Input
    IO0DIR |= (1<<16);                   // Configuring P0.10 as Output
    While (1)
    {
  If (!(IO1PIN & (1<<16)))               // Evaluates to True for a 'LOW' on P1.16
   {
     IO0CLR |= (1<<16);                 // drive P0.30 LOW, turn LED ON
   } else
  {
     IO0SET |= (1<<16);                 // drive P0.30 HIGH, turn LED OFF
  }
    }
  return 0;
}
```

OUTPUT:


ES_LAB - µVision4
1_LEDc
General Purpose Input/Output 1 (GPIO 1) - Slow Interface
Running with Code Size Limit: 32K
Load "C:\\Users\\Admin\\Documents\\ES_lab\\HEX\\1_LED_HEX.AXF"
*** Restricted Version with 32768 Byte Code Size Limit
*** Currently used: 576 Bytes (1%)


1_LEDc - ISIS Professional (Animating)
DEVICES
CERAMIC2N7
COMPIM
CRYSTAL
LED-RED
LPC2138
RES
ANIMATING: 00:00:07.750342 (CPU load 23%)

Application:

1. Indicator Lights.
2. Signal Indicators.
3. Alarms and Notifications.
4. Automotive Lighting.
5. Digital Displays.

Pre-Lab Viva Questions:

1. What is embedded C programming used for?
2. Is embedded C better than C?
3. List the advantage of embedded C.
4. Who is the father of embedded C?
5. How many keywords in embedded C?

Post-Lab Viva Questions:

1. What is the purpose of interfacing an LED with an ARM kit in your embedded C program?
2. How did you verify that the LED output on the ARM kit matches your program's expected behavior?
3. Discuss the role of the GPIO functions in your embedded C program.
4. Describe the importance of timing and delays in your LED control program.
5. What additional features or improvements could be made to enhance the functionality of your LED interfacing program?

RESULT:

Thus an embedded C program for interfacing an LED with an ARM microcontroller was implemented and demonstartion of the LED functionality for various light patterns was observed in the ARM kit.

Ex No:10 Date:	INTERFACING LCD DISPLAY WITH ARM USING KEIL C

OBJECTIVE

To design and implement an embedded C program for GPS navigation system to display the navigation data, directions, and location information to users by interfacing an LCD display with an ARM microcontroller using Keil C.

SYSTEM AND SOFTWARE TOOLS REQUIRED

1. ARM Evaluation Kit
2. ARM Development tools
3. LCD
4. KEIL C software

ALGORITHM

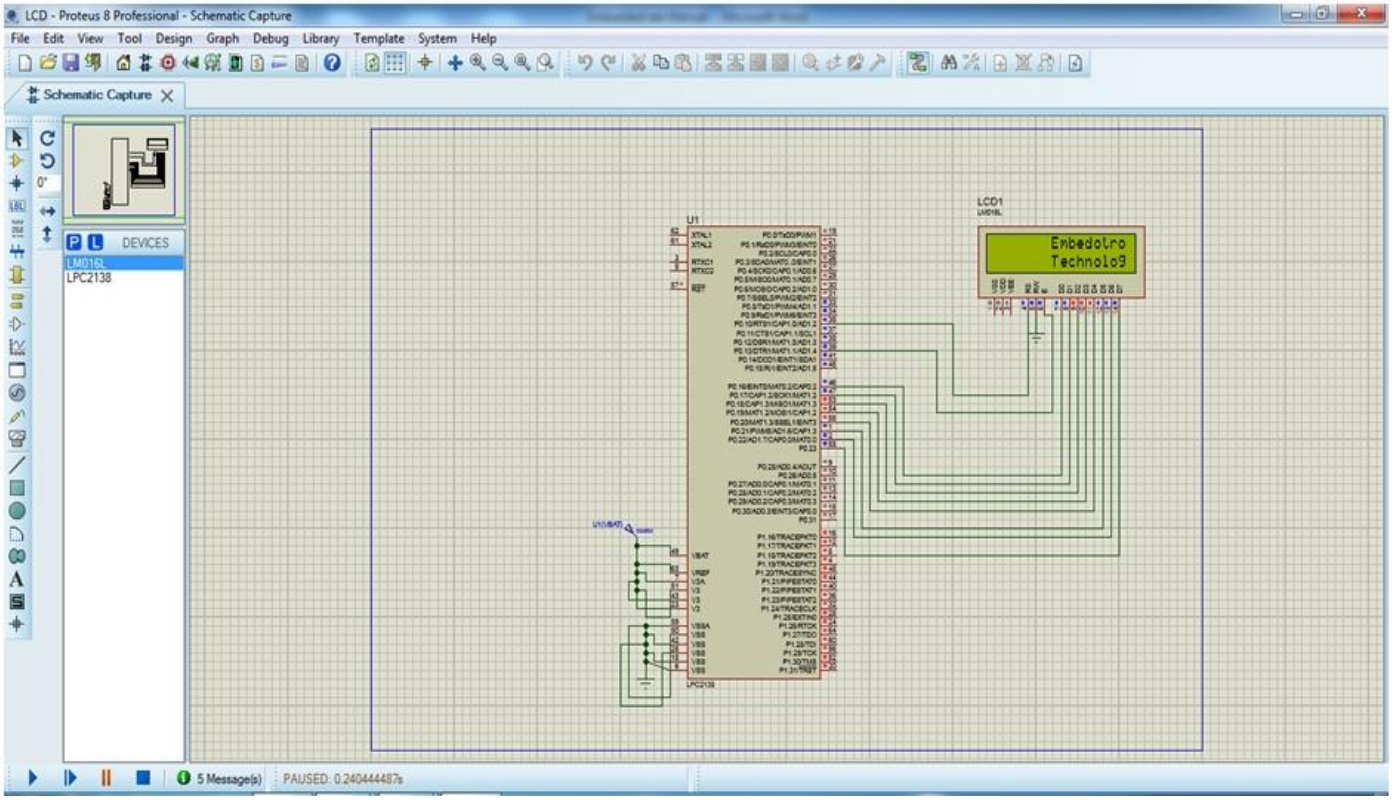
Step 1: Start the program
Step 2: Enable Input and Output Ports Step 3: Initialize timer
Step 4: Place the data for the LCD in the data bus
Step 5: Call delay
Step 6: Rotate the data for the LCD to switch on next number
Step 7: Use infinite loops for the continuous functioning of seven segment display
Step 8: Stop the program

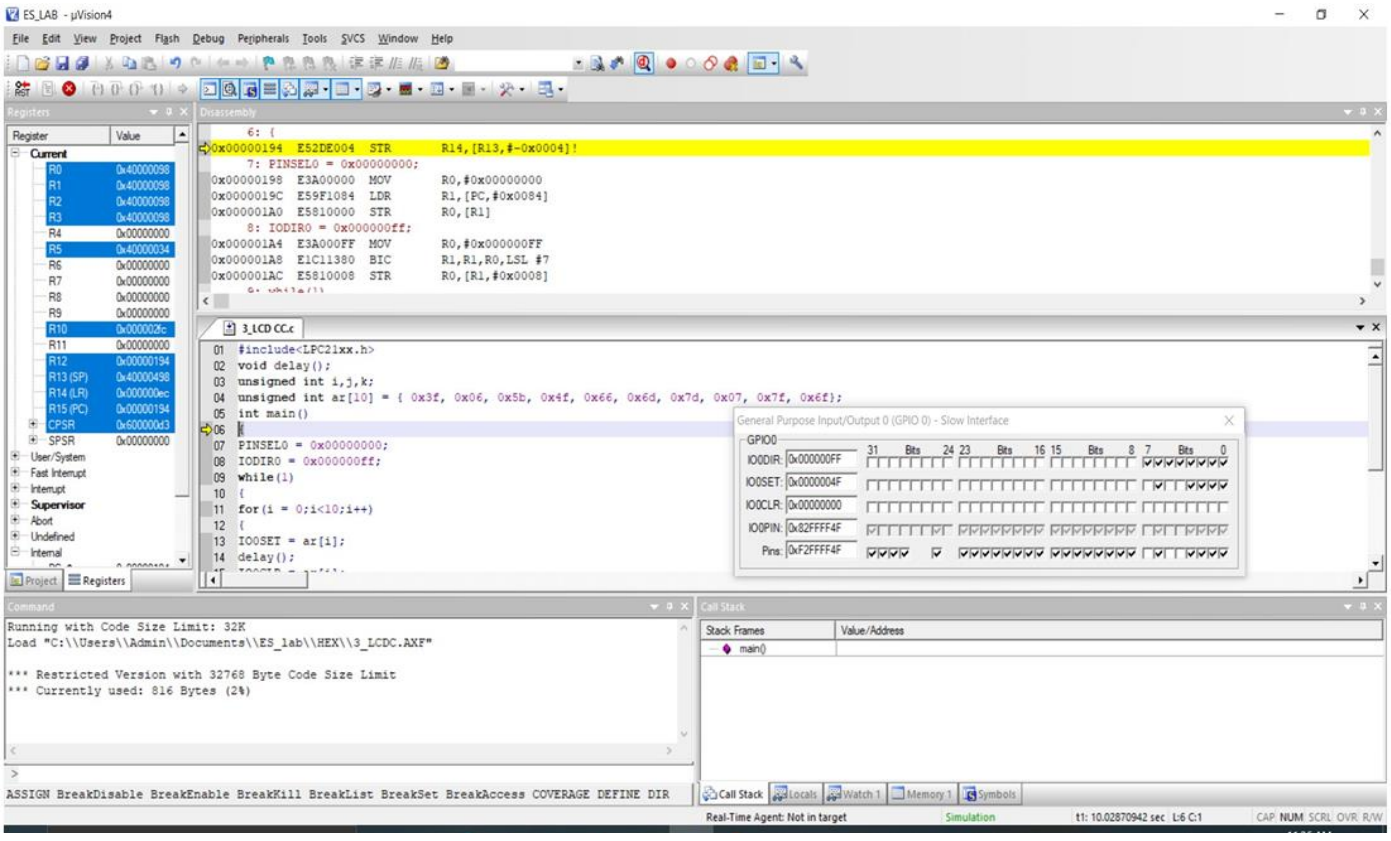
PROGRAM

```
#include <lpc214x.h>
 void initLCD(void);
//void enable(void);
void LCD_Write(unsigned int c);
void LCD_Cmd(unsigned int LCD_Cmd);
void delay(void);

int main(void)
{
      unsigned char ch[]="Embedotronics";
        unsigned char ch1[]="Technologies";
        unsigned int i,j,k,t;

   initLCD();

   for(i=0;ch[i]!='\0';i++)
         LCD_Write(ch[i]);

               LCD_Cmd(0xc3);
   for(j=0;ch1[j]!='\0';j++)
     {
               LCD_Write(ch1[j]);
```

```
        }
        while(1){
       for(k=0;k<16;k++)
       {
       LCD_Cmd(0x1c);
       for(t=0;t<300000;t++);
       }
       }
}

void initLCD(void)
{
   IO0DIR = 0x0FFFF00;
   delay(); //Initial Delay
   LCD_Cmd(0x38);
                LCD_Cmd(0x01);
                LCD_Cmd(0x0c);
                LCD_Cmd(0x83);
                LCD_Cmd(0x06);
}

void LCD_Write(unsigned int c)
{
   IO0PIN = (c<<16)|(1<<10);
   delay(); //Pulse Enable to process it
}

void LCD_Cmd(unsigned int LCD_Cmd)
{
   IO0PIN = (LCD_Cmd<<16)|(0<<10);
   delay(); //Pulse Enable to process it
}

void delay(void)
{
   int i=0,x=0;
        IO0PIN|=(1<<13);
   for(i=0; i<19999; i++)
   {
                        x++;
                }
                        IO0PIN&=~(1<<13);
```

OUTPUT:


LCD - Proteus 8 Professional - Schematic Capture
File Edit View Tool Design Graph Debug Library Template System Help
Schematic Capture
DEVICES
LM016L
LPC2138
LCD1
Embedotro
Technolog
5 Message(s)
PAUSED: 0.240444487s


ES_LAB - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
Register Value
Current
R0 0x40000098
R1 0x40000098
R2 0x40000098
R3 0x40000098
R4 0x00000000
R5 0x40000034
R6 0x00000000
R7 0x00000000
R8 0x00000000
R9 0x00000000
R10 0x000002fc
R11 0x00000000
R12 0x00000194
R13 (SP) 0x40000498
R14 (LR) 0x0000000ec
R15 (PC) 0x00000194
CPSR 0x600000d3
SPSR 0x00000000
User/System
Fast Interrupt
Interrupt
Supervisor
Abort
Undefined
Internal
Disassembly
6: {
0x00000194 E52DE004 STR R14,[R13,#-0x0004]!
7: PINSEL0 = 0x00000000;
0x00000198 E3A00000 MOV R0,#0x00000000
0x0000019C E59F1084 LDR R1,[PC,#0x0084]
0x000001A0 E5810000 STR R0,[R1]
8: IODIR0 = 0x000000ff;
0x000001A4 E3A000FF MOV R0,#0x000000FF
0x000001A8 E1C11380 BIC R1,R1,R0,LSL #7
0x000001AC E5810008 STR R0,[R1,#0x0008]
3_LCD CC.c
01 #include<LPC21xx.h>
02 void delay();
03 unsigned int i,j,k;
04 unsigned int ar[10] = { 0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f, 0x6f};
05 int main()
06 {
07 PINSEL0 = 0x00000000;
08 IODIR0 = 0x000000ff;
09 while(1)
10 {
11 for(i = 0;i<10;i++)
12 {
13 IO0SET = ar[i];
14 delay();
General Purpose Input/Output 0 (GPIO 0) - Slow Interface
GPIO0
IO0DIR: 0x000000FF
IO0SET: 0x0000004F
IO0CLR: 0x00000000
IO0PIN: 0x82FFFF4F
Pins: 0xF2FFFF4F
Command
Running with Code Size Limit: 32K
Load "C:\\Users\\Admin\\Documents\\ES_lab\\HEX\\3_LCDC.AXF"
*** Restricted Version with 32768 Byte Code Size Limit
*** Currently used: 816 Bytes (2%)
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE DEFINE DIR
Call Stack
Stack Frames Value/Address
main()
Call Stack Locals Watch 1 Memory 1 Symbols
Real-Time Agent: Not in target
Simulation
t1: 10.02870942 sec L:6 C:1
CAP NUM SCRL OVR R/W

Application:

1. Scientific Instruments
2. Display logged data and system status.
3. Industrial Equipment Monitoring
4. Temperature and Humidity Meters
5. Digital Clocks and Timers

Pre-Lab Viva Questions:

1. Which 3 pins are important in LCD interfacing?
2. What is the interfacing of LPC2148?
3. What is the principle of LCD?
4. List the three types of LCD.
5. What is LCD made of?

Post-Lab Viva Questions:

1. What are the key functions in your embedded C code that handle data and command transmission to the LCD?
2. List the role of LCD initialization routines in your embedded C program.
3. What challenges did you face when interfacing the LCD with the ARM kit, and how did you address them?
4. Provide the steps involved in initializing and configuring the LCD display in your embedded C program.
5. How do you handle different display modes (e.g., 8-bit vs. 4-bit) in your LCD interfacing program?

RESLUT

Thus an embedded C program for GPS navigation system to display the navigation data, directions, and location information for interfacing LCD display was designed and the output was observed with an ARM microcontroller using Keil C.

Ex No: 11 Date:	DESIGN SOLUTIONS FOR SMART HOME USING ARDUINO PROCESSOR

OBJECTIVE

To design a smart home system using an Arduino processor involves integrating various sensors, actuators, and communication modules to automate and control different aspects of the home environment.

SYSTEM AND SOFTWARE TOOLS REQUIRED

1. ARM Evaluation Kit
2. ARM Development tools
3. LCD
4. KEIL C software

ALGORITHM

Step 1: Start the program
Step 2: Define the system requirements
Step 3: Enable Input and Output Ports
Step 4: Read the data from the sensors and display in LCD in the data bus
Step 5: Call delay
Step 6: Develop the logic for automated response (turn on light, activating fan)
Step 7: Configure Wi-Fi connection and set up a web server.
Step 8: Implement endpoints to control home devices via a web interface.
Step 9: Stop the program

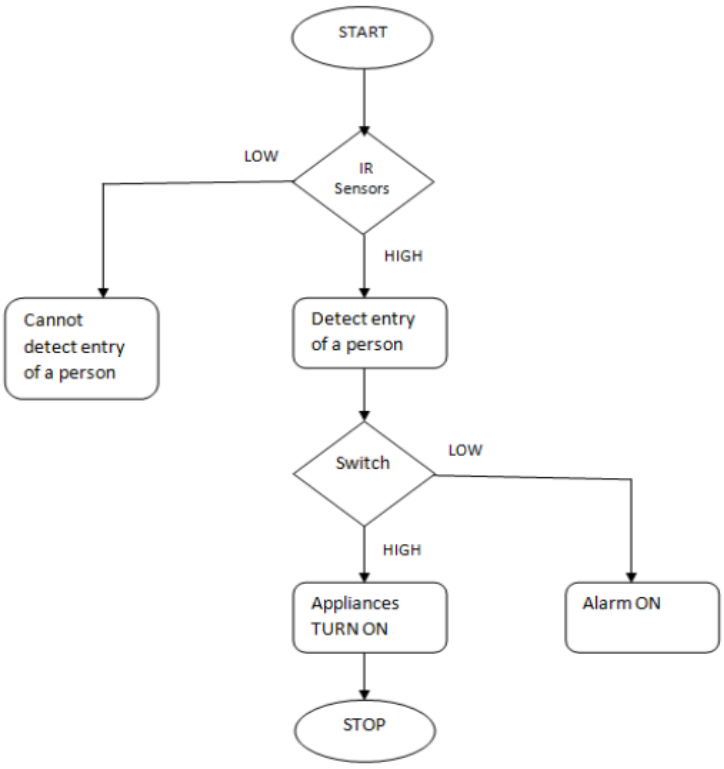
PROGRAM: 1 Lighting Control

```
int ldrPin = A0;        // LDR connected to analog pin A0
int relayPin = 8;       // Relay module connected to digital pin 8
int ldrValue;
void setup() {
  pinMode(relayPin, OUTPUT);
}

void loop() {
  ldrValue = analogRead(ldrPin);
  if (ldrValue < 300) { // Dark condition
    digitalWrite(relayPin, HIGH); // Turn on light
  } else {
    digitalWrite(relayPin, LOW); // Turn off light
  }
  delay(1000); // Delay for 1 second
}
```

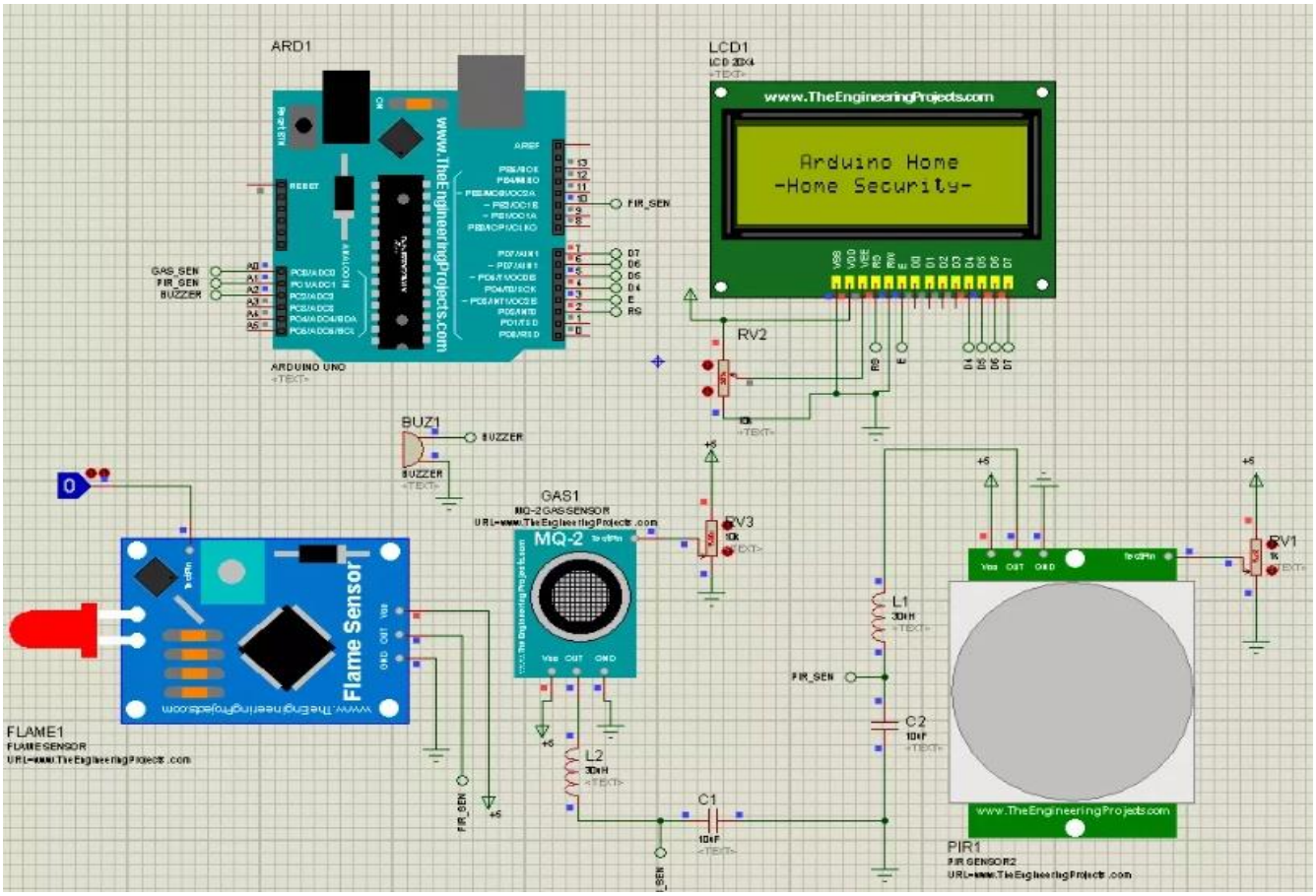
PROGRAM: 2 Temperature Control

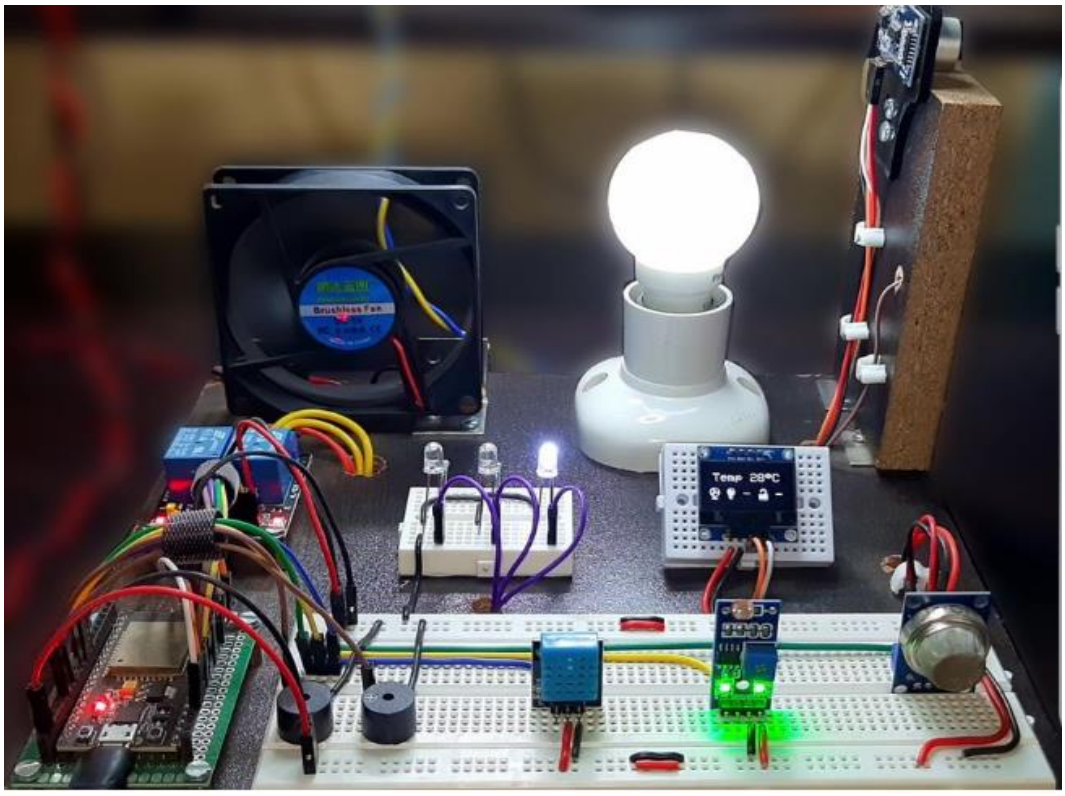
```
#include <DHT.h>
#define DHTPIN 2
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

int relayPin = 8; // Relay module connected to digital pin 8
float temperature;

void setup() {
  pinMode(relayPin, OUTPUT);
  dht.begin();
}

void loop() {
  temperature = dht.readTemperature();
  if (temperature > 25.0) { // If temperature is above 25°C
    digitalWrite(relayPin, HIGH); // Turn on fan
  } else {
    digitalWrite(relayPin, LOW); // Turn off fan
  }
  delay(2000); // Delay for 2 seconds
}
```

FLOWCHART:



OUTPUT:


ARD1
ARDUINO UNO
www.TheEngineeringProjects.com
LCD1
www.TheEngineeringProjects.com
Arduino Home
-Home Security-
RV2
BUZ1
BUZZER
GAS1
MQ-2
RV3
FLAME1
Flame Sensor
L1
L2
C1
C2
PIR1
RV1
FIR_SEN
GAS_SEN
BUZZER


Temp 28°C

Pre-Lab Viva Questions:

1. What are the primary goals of the smart home system you intend to design?
2. List the sensors and actuators are required for your smart home system.
3. What is the role of the Arduino in your smart home system?
4. List some o the home automations applications.
5. What do you by user interface which is needed to control or monitor the new features?

Post-Lab Viva Questions:

1. What were the major challenges or limitations encountered during the design process?
2. List the communication module effective for remote control and monitoring.
3. How will you integrate additional features or expand the functionality of your smart home system?
4. What were the major challenges or limitations encountered during the design and implementation of the smart home system?
5. How did you ensure the reliability and robustness of the system during operation?

RESLUT

Thus an embedded C program is developed to integrate additional features and functionality of the smart home automation system and the output is verified in the ARM kit.