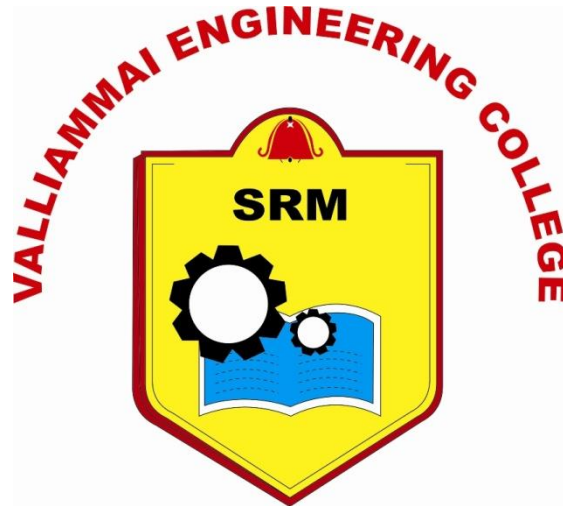


SRM VALLIAMMAI ENGINEERING COLLEGE

(An Autonomous Institution)

SRM NAGAR, KATTANKULATHUR – 603 203.

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION
ENGINEERING**



LAB MANUAL

1906605 MICROCONTROLLERS AND EMBEDDED LAB

III-YEAR VI-SEM

ACADEMIC YEAR: 2024-2025 (EVEN SEMESTER)

Prepared by

Dr. S. R. Preethi, Associate Professor /ECE

Dr. K. Durgadevi, Asst. Professor (O.G) /ECE

Ms. K. Arthi, Asst. Professor (O.G) /ECE

SRM VALLIAMMAI ENGINEERING COLLEGE

(An Autonomous Institution)
SRM Nagar, Kattankulathur -603 203

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

VISION OF THE INSTITUTE

“Educate to excel in social transformation”

To accomplish and maintain international eminence and become a model institution for higher learning through dedicated development of minds, advancement of knowledge and professional application of skills to meet the global demands.

MISSION OF THE INSTITUTE

- To contribute to the development of human resources in the form of professional engineers and managers of international excellence and competence with high motivation and dynamism, who besides serving as ideal citizen of our country will contribute substantially to the economic development and advancement in their chosen areas of specialization.
- To build the institution with international repute in education in several areas at several levels with specific emphasis to promote higher education and research through strong institute-industry interaction and consultancy.

VISION OF THE DEPARTMENT

To excel in the field of electronics and communication engineering and to develop highly competent technocrats with global intellectual qualities.

MISSION OF THE DEPARTMENT

- To educate the students with the state of art technologies to compete internationally, able to produce creative solutions to the society`s needs, conscious to the universal moral values, adherent to the professional ethical code
- To encourage the students for professional and software development career
- To equip the students with strong foundations to enable them for continuing education and research.

PROGRAMME OUTCOMES (POs)

- 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 modeling 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.

PROGRAMME SPECIFIC OUTCOMES (PSOs) of ECE DEPARTMENT

- PSO1:** Ability to apply the acquired knowledge of basic skills, mathematical foundations, and principles of electronics, modeling and design of electronics based systems in solving engineering Problems.
- PSO2:** Ability to understand and analyze the interdisciplinary problems for developing innovative sustained solutions with environmental concerns.
- PSO3:** Ability to update knowledge continuously in the tools like MATLAB, NS2, XILINIX and technologies like VLSI, Embedded, Wireless Communications to meet the industry requirements.
- PSO4:** Ability to manage effectively as part of a team with professional behavior and ethics.

OBJECTIVES:

The student should be made to:

- Write Assembly Language Program (ALP) for arithmetic and logical operations in 8086 and 8051
- Differentiate Serial and Parallel Interface.
- Understand the working of ARM Processor and study the interrupt performance.
- Enumerate programs to interface memory, I/O's with processor.
- Explore the concepts of Hardware of various microcontrollers to enable Programming and Interfacing of microcontroller.

LIST OF EXPERIMENTS: 8086 Programs using kits and MASM

1. Basic arithmetic and Logical operations.
2. Move a data block without overlap.
3. Code conversion, decimal arithmetic and Matrix operations.
4. String manipulations, sorting and searching.
5. Password checking, Print RAM size and system date.

LIST OF EXPERIMENTS: Peripherals and Interfacing Experiments using 8086 and ARM – 7 Processor.

6. Interfacing Traffic light controller.
7. Interfacing Stepper motor and Temperature sensor.
8. Implementing Zigbee protocol with ARM.
9. Interfacing Key board and LCD.
10. Interfacing LED and PWM.
11. Interfacing EPROM and Interrupt.
12. Analyze Serial interface and Parallel interface.
13. Interfacing ADC and DAC and Waveform Generation.

LIST OF EXPERIMENTS: 8051 Experiments using kit and MASM.

14. Program Basic arithmetic and Logical operations.
15. Implement Square and Cube program and Find 2's complement of a number.

TOTAL PERIODS: 60

OUTCOMES:

On completion of this laboratory course, the student would be able to,

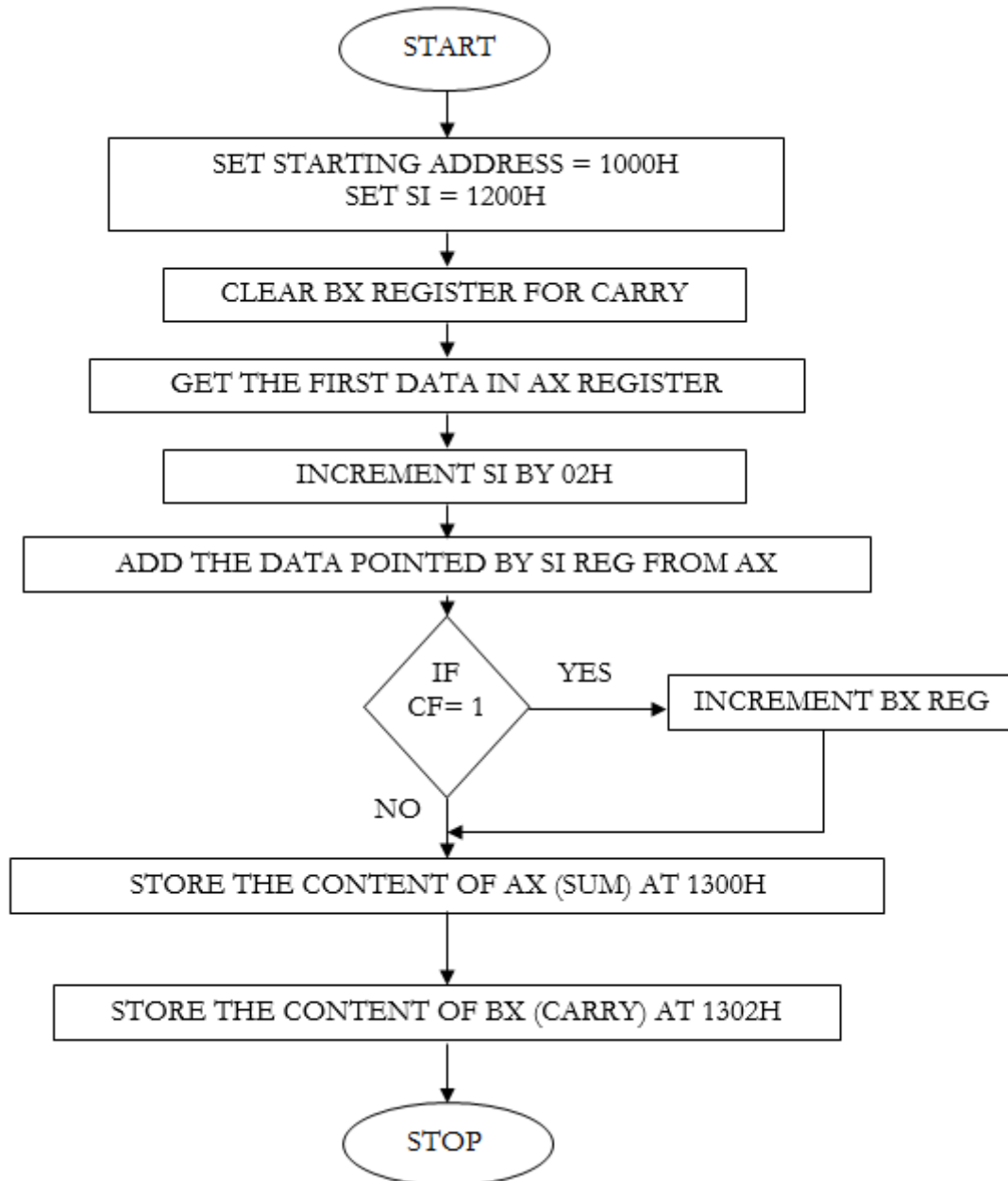
- Write ALP Programs for Arithmetic operations and Logical operations.
- Express the programming logics for code conversion and acquire knowledge on ADC and DAC.
- Interface different I/O's with processor and Generate waveforms using 8086 and ARM processors.
- Execute microcontroller programs in 8051.
- Formulate a mini Project using Embedded System.

CONTENTS

Sl. No.	Name of the Experiments	Signature
CYCLE - I		
1	Basic arithmetic and Logical operations	
2	Move a data block without overlap	
3	Code conversion, decimal arithmetic and Matrix operations.	
4	String manipulations, sorting and searching	
5	Password checking, Print RAM size and system date	
CYCLE – II		
6	Interfacing Traffic light control	
7	Interfacing Stepper motor control and Temperature sensor	
8	Implementing Zigbee protocol with ARM	
9	Interfacing Key board and LCD	
10	Interfacing LED and PWM	
11	Interfacing EPROM and Interrupt.	
12	Analyze Serial interface and Parallel interface	
13	Interfacing ADC and DAC and Waveform Generation	
CYCLE – III		
14	Program Basic arithmetic and Logical operations	
15	Implement Square and Cube program and Find 2's complement of a number	
TOPIC BEYOND SYLLABUS		
16	Square wave generation using 8051	

CYCLE I
8086 PROGRAMS

Flow Chart for Addition of Two Numbers:



Ex. No. 1

Date:

PROGRAMS FOR BASIC ARITHMETIC AND LOGICAL OPERATIONS

Objective:

To write an Assembly Language Program (ALP) to perform basic Arithmetic and Logical Operations

- (a) Addition of two numbers
- (b) Subtraction of two numbers
- (c) Multiplication of two numbers
- (d) Division of two numbers
- (e) Logical operation

(A) ADDITION OF TWO 16 BIT NUMBERS

Description:

To perform addition in 8086, one of the data should be stored in a register and another data can be stored in register / memory. After addition the sum will be available in the destination register / memory. The sum of two 16-bit data can be either 16 bits (sum only) or 17 bits (sum and carry). The destination register / memory can accommodate only the sum and if there is a carry the 8086 will indicate by setting carry flag. Hence one of the register is used for the account of carry.

Algorithm:

1. Start the program.
2. Set the origin as 1000H.
3. Store the 1st data in AX register.
4. Clear BX register pair for carry.
5. Set SI to 1202H to point the second data.
6. Add the content in AX with data pointed by SI register.
7. If carry occurs, increment BX register by one.
8. Move the content of AX to 1300H.
9. Move the content of BX to 1302H.
10. End of segment.
11. Stop the program

PROGRAM

Label	Program	Comments
	ORG 1000H	Set starting address as 1000H.
	MOV BX, 0000H	Initialize BX to 0000H
	MOV SI, 1200H	Move immediate data to SI
	MOV AX, [SI]	Move content of SI to AX
	ADD SI, 02H	ADD SI with immediate data.
	ADD AX, [SI]	Add content of SI with AX register
	JNC Next	Jump if no carry to loop
	INC BX	Increment BX register
Next:	MOV DI, 1300H	Move immediate data to DI.
	MOV [DI], AX	Move AX to DI.
	ADD DI, 02H	ADD DI with immediate data
	MOV [DI], BX	Move BX to DI
	HLT	

Example 1:

Manual Calculation:

With Carry

Input:

1200: 46H
 1201: B6H [Addend]
 1202: D3H
 1203: 98H[Augend]

Output:

1300: 19H
 1301: 4FH [Sum]
 1302: 01H
 1303: 00H [Carry]

Example 2:

Without Carry

Input:

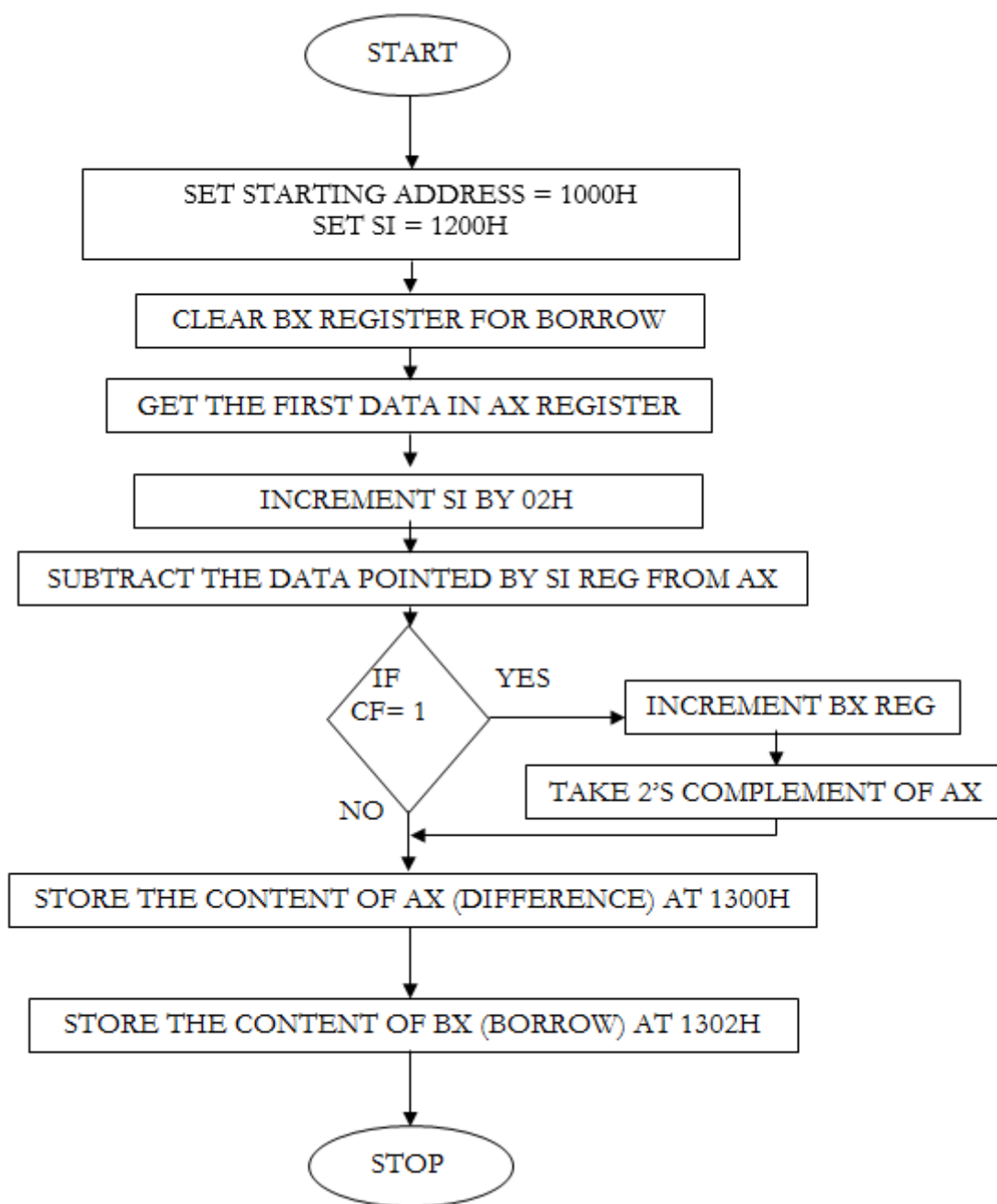
1200: 34H
 1201: 44H [Addend]
 1202: 24H
 1203: 24H [Augend]

Output:

1300: 58H
 1301: 68H [Sum]
 1302: 00H

1303: 00H [Carry]

Flow Chart of Subtraction of Two Numbers:



(B) SUBTRACTION OF TWO 16 BIT NUMBERS

Description:

To perform subtraction in 8086 one of the data should be stored in register and another data should be stored in register or memory. After subtraction the result will be available in destination register/memory. The 8086 will perform 2's complement subtraction and then complement the carry. Therefore, if the result is negative then carry flag is set and the destination register/memory will have 2's complement of the result. Hence one of the registers is used to account for sign of the result. To get the magnitude of the result again take 2's complement of the result.

Algorithm:

1. Start the program.
2. Set the starting address as 1000H.
3. Set the SI register to 1200H address.
4. Move the 16-bit data to AX register pair.
5. Increment the SI register to 1202.
6. Get the second data.
7. Move this second value to BX register.
8. Subtract the content pointed by SI from AX and store result in AX.
9. If carry occurs go to step 13.
10. Increment BX register, then perform inversion operation to AX register.
11. Increment AX register.
12. Move the resultant to DI register.
13. Display the output.
14. End of segment.
15. Stop the program.

PROGRAM

Label	Program	Comments
	ORG 1000H	Set starting address as 1000H
	MOV BX, 0000H	Move immediate data to BX register.
	MOV SI, 1200H	Move immediate data to SI
	MOV AX, [SI]	Move contents of SI to AX
	ADD SI, 02H	Increment SI by 02H
	SUB AX, [SI]	Move contents of SI to AX
	JNC Next	Jump if no carry loop
	INC BX	Increment BX
	NOT AX	Perform NOT operation of AX
	INC AX	Increment AX register
Next:	MOV DI, 1300H	Move immediate data to DI.
	MOV [DI], AX	Move AX to DI.
	ADD DI, 02H	Increment DI by 02H
	MOV [DI], BX	Move BX to DI
	HLT	

Example 1:

With Borrow

Input:

1200: 03H

1201: 00H (minuend)

1202: 05H

1203: 00H (subtrahend)

Manual Calculation:

Output:

1300: 02H
1301: 00H (Difference)
1302: 01H
1303: 00H (Borrow)

Example 2:

Without Borrow

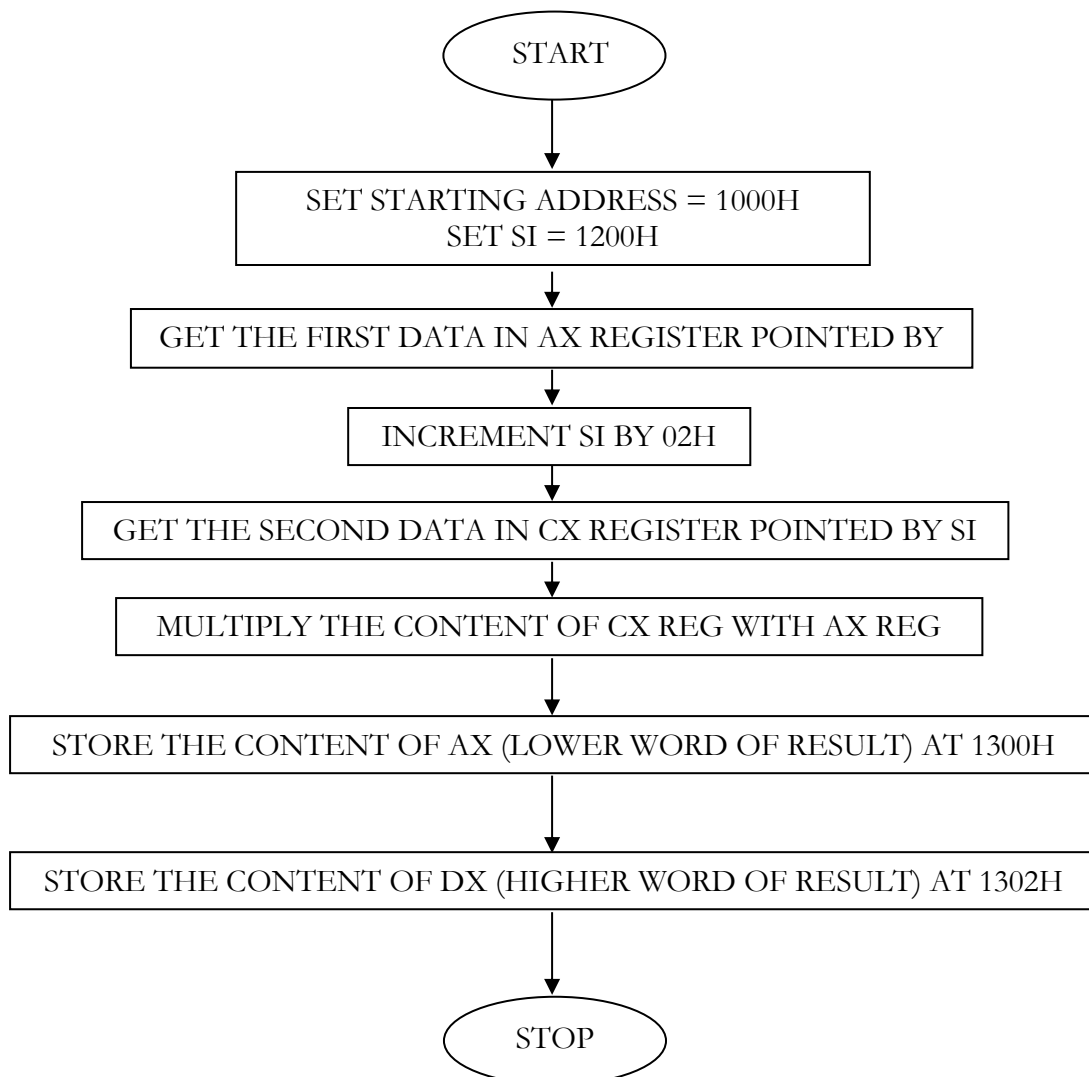
Input:

1200: 31H
1201: 82H (minuend)
1202: 06H
1203: 34H (subtrahend)

Output:

1300: 2BH
1301: 4EH (Difference)
1302: 00H
1303: 00H (Borrow)

Flow Chart for Multiplication of Two Numbers:



(C) MULTIPLICATION OF TWO 16 BIT NUMBERS

Description:

To perform multiplication in 8086 processors one of the data should be stored in AX register and another data can be stored in register/memory. After multiplication the product will be in AX [lower word] and DX register [Higher word].

Algorithm:

1. Start the program
2. Set the starting address as 1000H
3. Set the SI register to point the location 1200H.
4. Set the DI register to point the location 1300H.
5. Move the 16-bit data pointed by SI to AX register
6. Move this data to BX register
7. Increment SI register to 1202 and get the second data in AX register
8. Multiply the data in AX with BX register
9. Store the data in DX [higher word] and AX [lower word] addressed by DI register.
10. Display the result
11. End of segment
12. Stop the program

PROGRAM

Label	Program	Comments
	ORG 1000H	Set starting address as 1000H.
	MOV SI, 1200H	Move immediate data to SI
	MOV AX,[SI]	Move contents of SI to AX
	ADD SI,02H	Increment SI value to 02H
	MOV BX, [SI]	Move contents of SI to BX
	MUL BX	Multiply BX with AX
	MOV DI, 1300H	Move immediate data to DI
	MOV [DI], AX	Move AX to DI register
	MOV DI, 1302H	Move immediate data to DI
	MOV [DI], DX	Move DX to DI register
	HLT	

Example:

Manual Calculation:

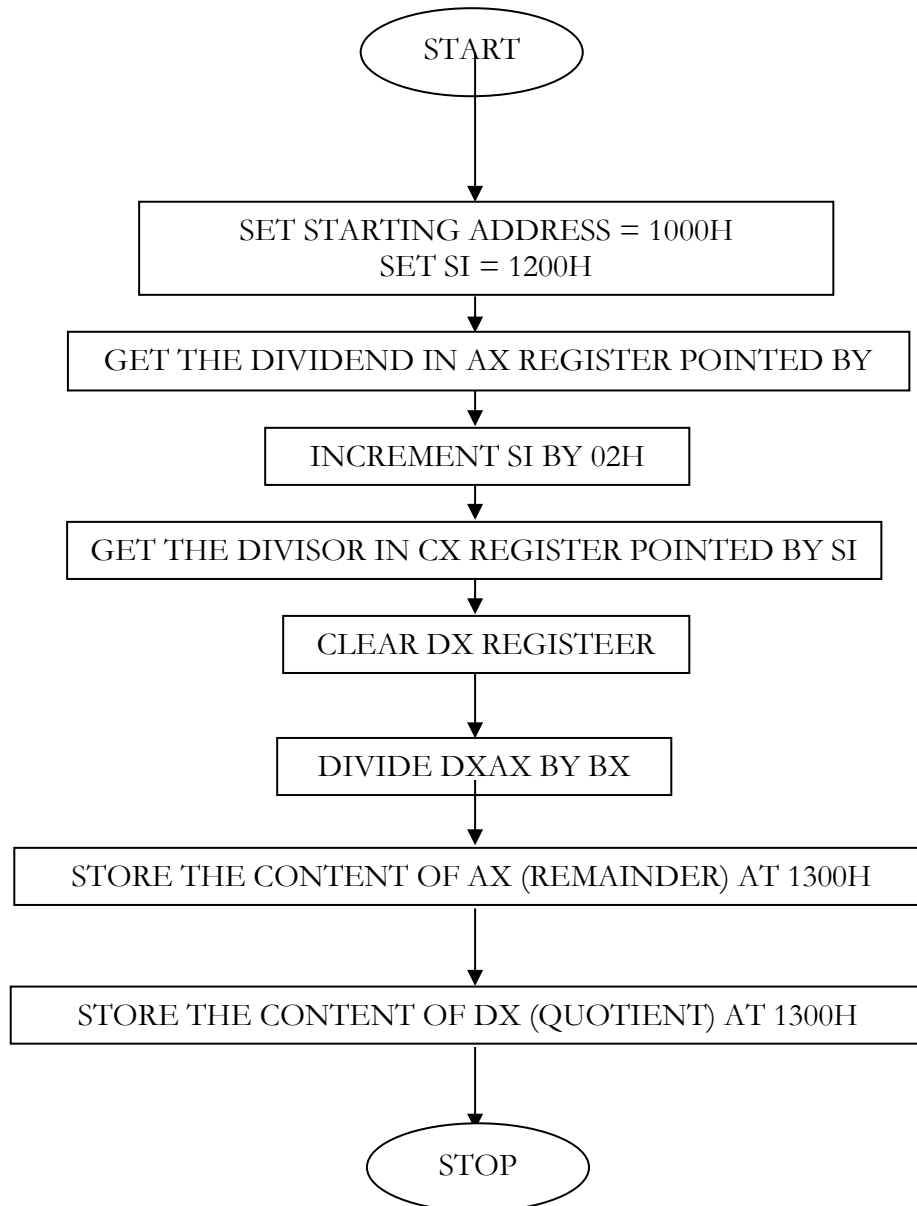
Input:

1200: 02H
1201: 06H (Multiplicand)
1202: 02H
1203: 06H (Multiplier)

Output:

1300: 04H
1301: 18H (Lower word of the Product)
1302: 24H
1303: 00H (Higher word of the Product)

Flow Chart for Division of Two Numbers:



(D) DIVISION OF TWO NUMBERS

Description:

To perform division in 8086 processor, the 16 bit dividend should be stored in AX and DX register (The lower word in AX and Upper word in DX). The 16 bit divisor can be stored in register / memory. After division the quotient will be in AX register and the remainder will be in DX register.

Algorithm:

1. Start the program
2. Set the origin as 1000H
3. Set SI as 1200H.
4. Clear DX register for 16 bit dividend. For 16 bit dividend higher word is zero.
5. Load the lower word of dividend in AX register
6. Increment SI by 02H. Load the divisor in BX register.
7. Perform division of data in DX AX by BX
8. Set DI as 1300H
9. Store the quotient in AX register at the location pointed by DI register.
10. Set DI as 1302H
11. Store the remainder in DX register at the location pointed by DI register.
12. Display the result, End of Segment
13. Stop the program

PROGRAM

Label	Program	Comments
	ORG 1000H	Set starting address as 1000H.
	MOV SI, 1200H	Move immediate data to SI
	MOV AX,[SI]	Move contents of SI to AX
	ADD SI,02H	Add 02H to SI
	MOV BX, [SI]	Move contents of SI to BX
	MOV DX, 0000H	Initialize DX to 0000H
	DIV BX	Divide DXAX by BX
	MOV DI, 1300H	Move immediate data to DI
	MOV [DI], AX	Store the quotient
	MOV DI, 1302H	Move immediate data to DI
	MOV [DI], DX	Store the remainder
	HLT	

Example:

Manual Calculation:

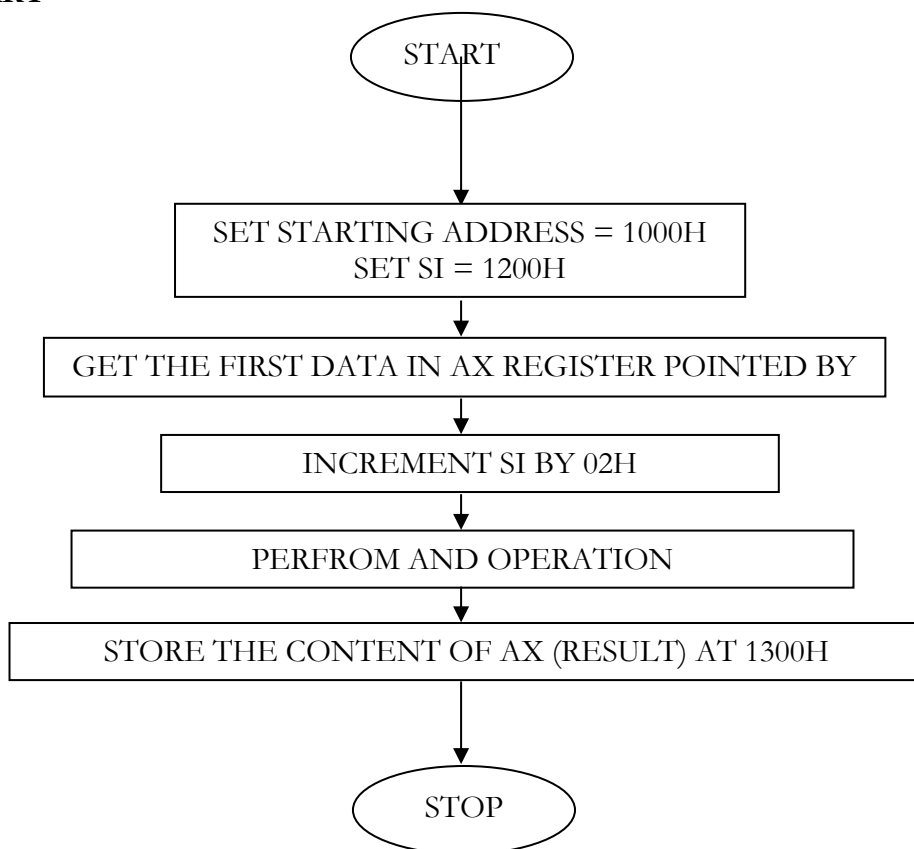
Input:

1200: 06H
1201: 06H (Dividend)
1202: 03H
1203: 03H (Divisor)

Output:

1300: 02H
1301: 00H (Quotient)
1302: 00H
1303: 00H (Remainder)

FLOWCHART



(E) LOGICAL OPERATIONS OF 16 BIT NUMBERS

Description:

The two values from memory are logically AND then the result is stored in memory.

Algorithm:

1. Start the program and Set the origin as 1000H
2. Set SI as 1200H.
3. Get the first data in AX – reg
4. Increment SI to point next data
5. Perform AND operation of the data
6. Store the result in memory
7. Stop the program

PROGRAM

Label	Program	Comments
	ORG 1000H MOV SI,1200H MOV AX,[SI] ADD SI,02H AND AX,[SI] MOV DI,1300H MOV [DI],AX HLT	Set starting address as 1000H. Initialize SI Get the first data in AX – reg Increment SI to point next data Perform AND operation of two data Store the result in memory

Example:

Manual Calculation:

Input

1200: 01H
1201:01H
1202:00H
1203:00H

Output

1300:00H
1301:00H

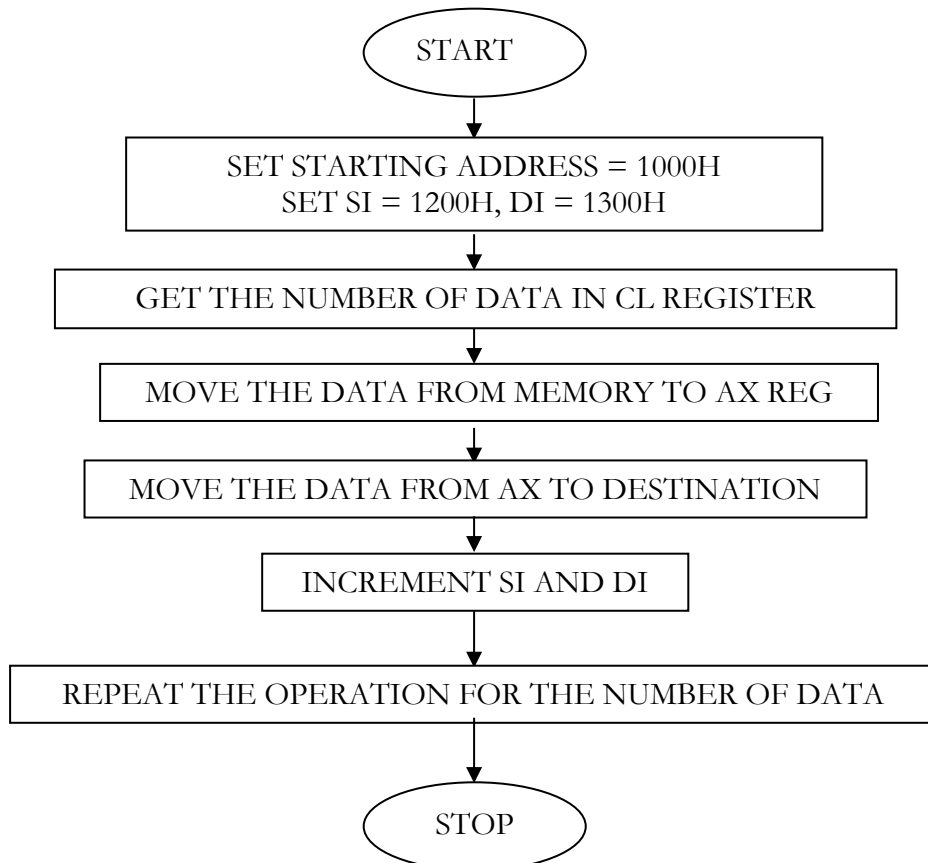
REVIEW QUESTIONS:

1. Write the size of the data bus of 8086.
2. Write the size of the address bus of 8086.
3. What is meant by physical addressing in 8086?
4. What are the other possibilities of writing ADD, SUB and MUL instructions in other addressing modes?
5. What is the purpose of BIU& EU?

Result:

Thus the program for arithmetic and logic operation was written and executed.

Flow Chart to Move a Block of Data without Overlap:



Ex. No. 2

Date:

MOVE A DATA BLOCK WITHOUT OVERLAP

Objective:

To write an 8086 ALP to move a block of data from source to destination without overlap

Description:

The block of data to be moved from one location (source) to another location (destination) in memory. The source and destination of memory is pointed by SI and DI respectively. The size of the block is stored in CL register. The data from source are moved to register and then back to destination location. The steps are repeated till the value of CL register is Zero.

Algorithm:

1. Start the program.
2. Set the starting address as 1000H.
3. Set the SI register to 1200H address.
4. Set the DI register to 1300H address.
5. Set the CL register to hold the number of data to be moved.
6. Move the 16-bit data from memory pointed by SI to AX register pair.
7. Move the 16-bit from AX register to memory pointed by DI.
8. Increment the SI register by 02H.
9. Increment the DI register by 02H.
10. Repeat steps 6 to 9 till the cl value is zero
11. Stop the program.

PROGRAM

Label	Program	Comments
Next:	ORG 1000H MOV SI, 1200H MOV DI,1300H MOV CL,05H MOV AX,[SI] MOV [DI],AX ADD SI,02H ADD DI, 02H LOOP Next HLT	Set starting address as 1000H. Initialise SI to 1200 Initialise DI to 1300 Initialise CL for number of data

Example:

Manual Calculation:

Input:

1200: 05H
1201: 03H
1202: 02H
1203: 01H
1204: 00H

Output:

1300: 05H
1301: 03H
1302: 02H
1303: 01H
1304: 00H

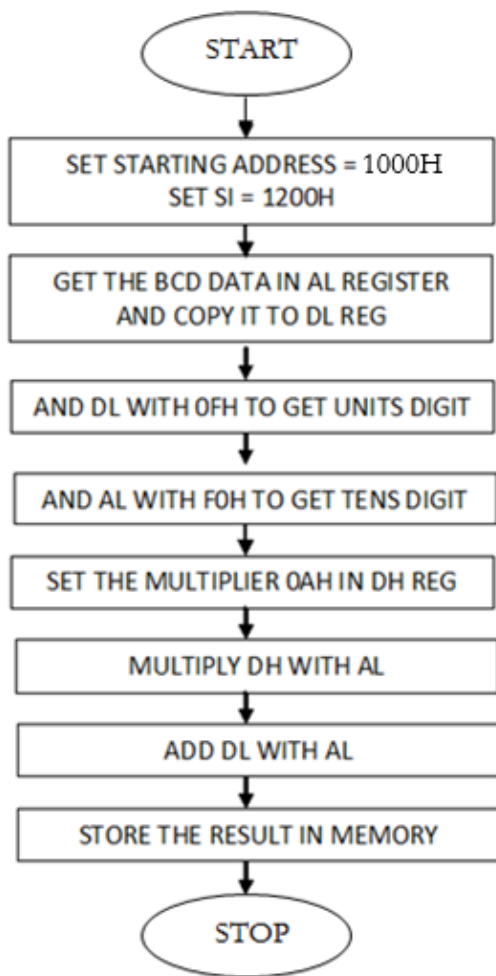
REVIEW QUESTIONS:

1. List out the Flag manipulation instruction.
2. Give the differences between JUMP and LOOP instruction
3. List out the advantages of using Direct Memory Access (DMA).
4. What is meant by Maskable interrupts& Non-Maskable interrupts?
5. What is the Maximum clock frequency in 8086?

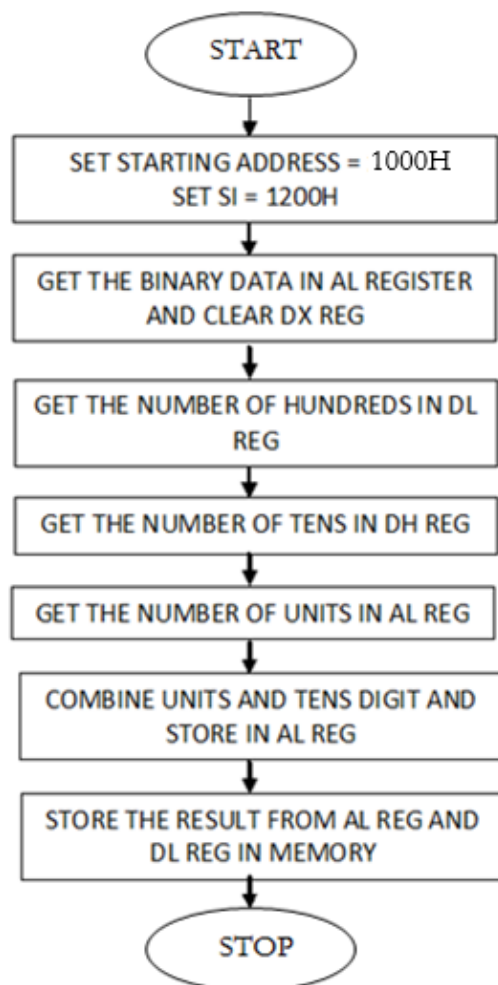
Result:

Thus the program for moving a block of data without overlap was written and executed.

Flow Chart for Binary to BCD Conversion



Flow Chart for BCD to Binary Conversion



Ex. No. 3**Date:****CODE CONVERSION, DECIMAL ARITHMETIC & MATRIX OPERATIONS****Objective:**

To write an Assembly Language Program (ALP) to perform the following operations

- (a) Code Conversion
 - BCD to Binary
 - Binary to BCD
- (b) Decimal Arithmetic
 - BCD Addition
 - BCD Subtraction
- (c) Matrix Operations
 - Matrix Addition
 - Matrix Multiplication

(A) CODE CONVERSION – BCD to Binary**Description:**

The 2 –digit BCD data will have units digits and tens digits. When the tens digit is multiplied by 0A H and the product is added to units digit, the result will be in binary, because the microprocessor will perform binary arithmetic. In order to separate the units and tens digit, masking technique is used.

Algorithm:

1. Start the program.
2. Set the origin as 1000H.
3. Get the BCD data in AL register
4. Copy the BCD data in DL register
5. Logically AND DL with 0F to mask upper nibble and get the units digit in DL
6. Logically AND AL with F0 to mask lower nibble and get the tens digit in AL
7. Rotate the content of AL register 4 times in order to change upper nibble as lower nibble.
8. Set the multiplier 0A H in DH register.
9. Multiply AL with DH register, the product will be in AL register.
10. Add the units digit in DL register to the product in AL register
11. Save the binary digit (AL) in memory
12. Stop the program.

PROGRAM

Label	Program	Comments
	ORG 1000H	Set starting address as 1000H.
	MOV SI, 1200H	Initialize SI
	MOV AL,[SI]	Move the BCD data in AL
	MOV DL,AL	Copy the BCD data in DL
	AND DL,0F	AND DL with 0F

AND AL,0F0	AND AL with F0
MOV CL,04	
ROR AL,CL	Rotate AL for 4 – times
MOV DH,0A	Move 0A to DH
MUL DH	Multiply DH with AL
ADD AL,DL	Add AL with DL
MOV DI,1201H	
MOV [DI],AL	Store the result in memory
HLT	

Example:

Manual Calculation:

Input:

1200: 85H [BCD data]

Output:

1201: 55H

Result:

Thus the program for BCD to Binary conversion was successfully executed.

CODE CONVERSION – BINARY TO BCD

Description:

The maximum value of 8 bit binary is FFH. The BCD equivalent is 256. Hence when an 8 – bit binary is converted into BCD, the BCD data will have hundreds, tens and units digit. So two counters are used to count hundreds and tens. The tens and units digit are added and stored in a memory location and the hundreds digit is stored in the next location.

Algorithm:

1. Start the program.
2. Set the origin as 1000H.
3. Get the binary data in AL register
4. Clear DX register for storing Hundreds and tens
5. Compare AL with 64H (100 in decimal)
6. Check carry flag. If CF = 1, then go to step 10, else go to next step
7. Subtract 64H from AL register
8. Increment Hundreds register (DL)
9. Go to Step 5
10. Compare AL with 0AH (10 in decimal)
11. Check carry flag. If CF = 1, then go to step 15, else go to next step
12. Subtract 0AH from AL register
13. Increment Tens register (DH)
14. Go to step 10
15. Rotate the content of DH four times
16. Add DH to AL to combine tens and Units digit
17. Save AL and DL in memory.
18. Stop the program

PROGRAM

Label	Program	Comments
	ORG 1000H	Set starting address as 1000H.
	MOV SI, 1200H	Initialize SI
	MOV AL,[SI]	Move the binary data in AL
	MOV DX,0000H	Clear the counter
HUND:	CMP AL, 64H	To count number of hundreds
	JC TEN	
	SUB AL,64H	
	INC DL	
	JMP HUND	
TEN:	CMP AL,0AH	To count number of tens
	JC UNIT	
	SUB AL,0AH	
	INC DH	
	JMP TEN	
UNIT:	MOV CL,04	
	ROL DH,CL	
	ADD AL,DH	Add tens and units
	MOV DI,1201H	
	MOV [DI],AL	Store in memory
	INC DI	
	MOV [DI],DL	
	HLT	

Example:

Input:

1200: 55H [Binary data]

Output:

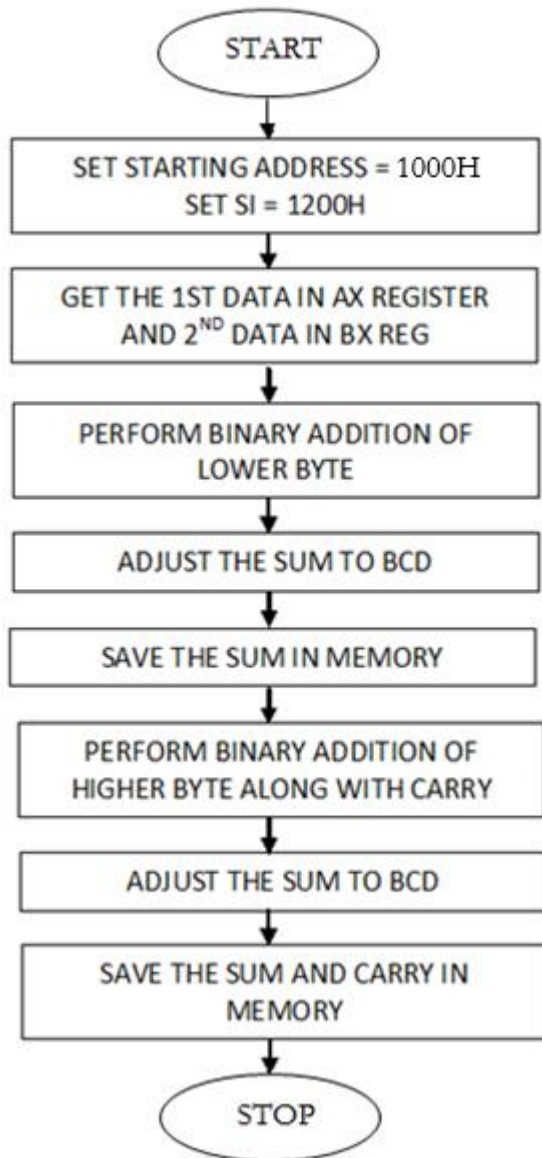
1201:85H

Manual Calculation:

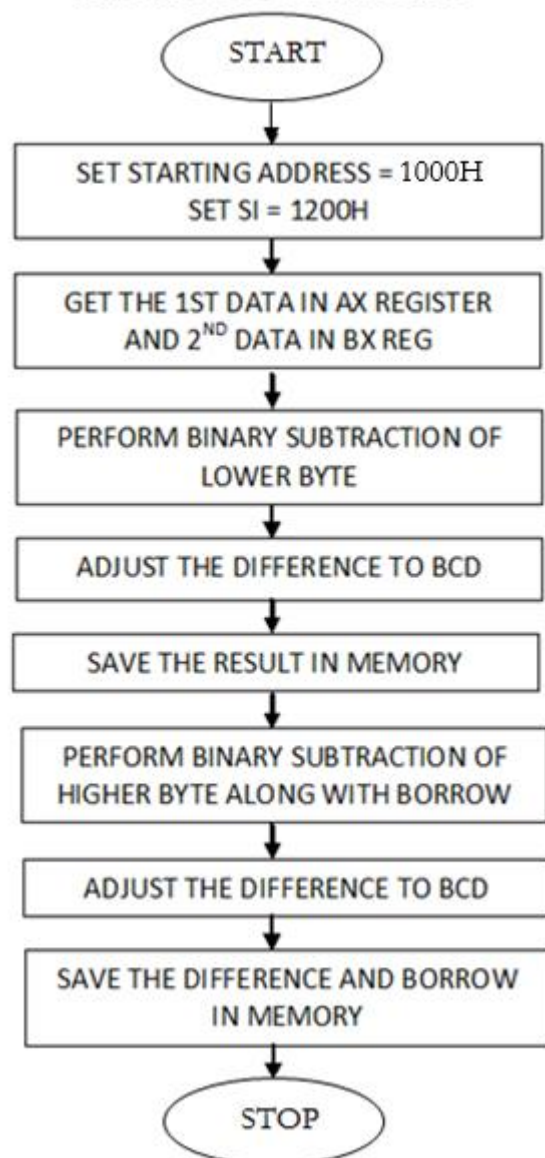
Result:

Thus the program for Binary to BCD conversion was successfully executed.

Flow Chart for BCD Addition



Flow Chart for BCD Subtraction



DECIMAL ARITHMETIC – BCD ADDITION

Description:

The binary addition is performed and then the sum is corrected to get the result in BCD. If the sum of the lower nibble exceeds 9 or if there is auxiliary carry then 6 is added to the lower nibble. if the sum of the upper nibble exceeds 9 or if there is a carry then 6 is added to upper nibble. These conversions are taken care by DAA instruction.

Algorithm:

1. Start the program.
2. Set the origin as 1000H.
3. Initialise SI to 1200H
4. Clear the CL register for Carry
5. Load the first data in AX reg and second data in BX reg.
6. Perform Binary addition of lower byte
7. Adjust the sum of lower bytes to BCD
8. Save the sum in memory.

9. Perform Binary addition of Higher byte along with carry from lower byte.
10. Adjust the sum of higher bytes to BCD
11. Save the sum in memory
12. Save the carry in memory
13. Stop the program.

PROGRAM

Label	Program	Comments
	ORG 1000H	Set starting address as 1000H.
	MOV SI, 1200H	Initialize SI
	MOV CL,00H	Clear CL register for carry
	MOV AX,[SI]	Get the 1 st number in AX reg
	MOV BX,[SI+2]	Get the 2 nd number in BX reg
	ADD AL,BL	Add the lower nibble
	DAA	Decimal adjust for BCD
	MOV DL,AL	
	MOV AL,AH	
	ADC AL,BH	Add the higher nibble with carry
	DAA	Decimal adjust for BCD
	MOV DH,AL	
	JNC AHEAD	Check for Carry
	INC CL	
AHEAD:	MOV DI,1204H	
	MOV [DI],DX	Store the result in memory
	MOV [DI+2],CL	
	HLT	

Example:

Manual Calculation:

Input:

```

1200: 01H   [ 1st data – BCD ]
1201: 04H
1202: 08H   [ 2nd data – BCD ]
1203: 02H

```

Output:

```

1204: 09H
1205: 06H

```

Result:

Thus the program for BCD addition was successfully executed.

DECIMAL ARITHMETIC – BCD SUBTRACTION

Description:

The binary subtraction is performed and then the difference is corrected to get the result in BCD. If the difference of the lower nibble exceeds 9 or if there is auxiliary carry then 6 is subtracted from the lower nibble. if the difference of the upper nibble exceeds 9 or if there is a carry then 6 is subtracted from upper nibble. This conversion is taken care by DAS instruction.

Algorithm:

1. Start the program.
2. Set the origin as 1000H.
3. Initialise SI to 1200H
4. Clear the CL register for borrow
5. Load the first data in AX reg and second data in BX reg.
6. Perform Binary subtraction of lower byte
7. Adjust the difference of lower bytes to BCD
8. Save the result in memory.
9. Perform Binary subtraction of Higher byte along with borrow from lower byte.
10. Adjust the difference of higher bytes to BCD
11. Save the difference in memory
12. Save the borrow in memory
13. Stop the program.

PROGRAM:

Label	Program	Comments
	ORG 1000H	Set starting address as 1000H.
	MOV SI, 1200H	Initialize SI
	MOV CL,00H	Clear CL register for borrow
	MOV AX,[SI]	Get the 1 st number in AX reg
	MOV BX,[SI+2]	Get the 2 nd number in BX reg
	SUB AL,BL	Subtract the lower nibble
	DAS	Decimal adjust for BCD
	MOV DL,AL	
	MOV AL,AH	
	SBB AL,BH	Subtract the higher nibble with Borrow
	DAS	Decimal adjust for BCD
	MOV DH,AL	
	JNC AHEAD	Check for Borrow
	INC CL	
AHEAD:	MOV DI,1204H	
	MOV [DI],DX	Store the result in memory
	MOV [DI+2],CL	
	HLT	

Example:

Input:

1200: 18[1st data – BCD]
1201: 04
1202: 09[2nd data – BCD]

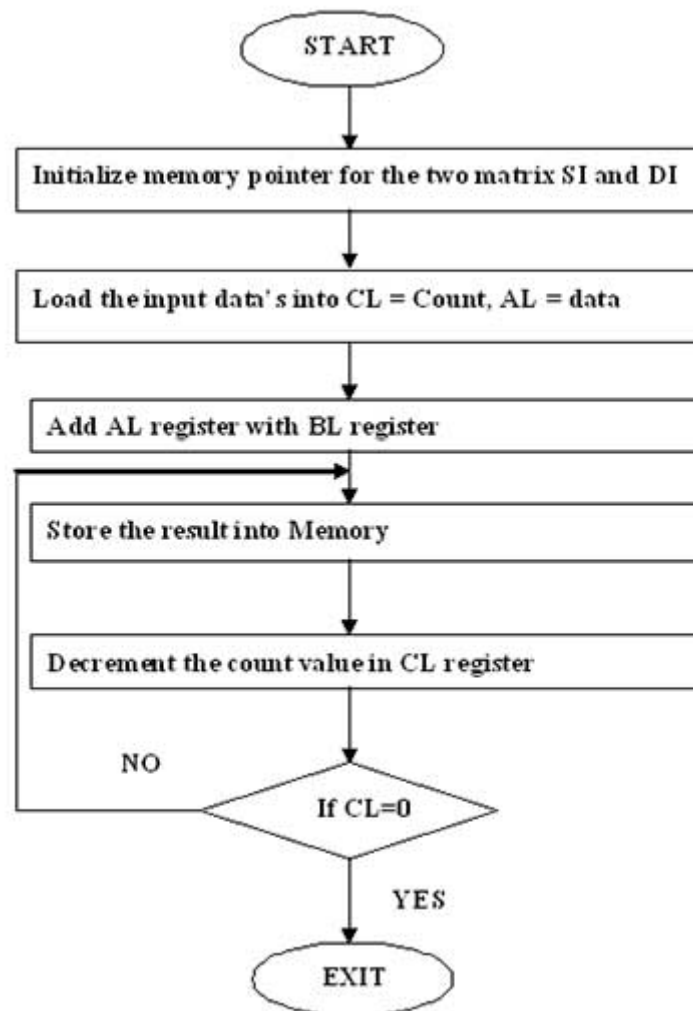
Output:

1204: 09
1205: 02
1203: 02

Result:

Thus the program for BCD subtraction was successfully executed.

Flow Chart for Matrix Addition:



MATRIX ADDITION

Description:

The matrix addition is performed by loading the size of the matrix in CL reg and then adding the individual elements of the matrix.

Algorithm:

1. Start the program.
2. Set the origin as 1000H.
3. Initialize the pointer to memory for data and result.
4. Load CL with count.
5. Add two matrices by each element.
6. Process continues until CL is 0.
7. Store the result into Memory.
8. Stop the program.

PROGRAM

LABEL	PROGRAM	COMMENTS
	MOV CL, 09	Initialize 09 into CL register
	MOV SI, 2000	Load 2000 into SI for 1 st matrix
	MOV DI, 3000	Load 3000 into DI for 2 nd matrix
NEXT	MOV AL, [SI]	Load AL with data of first matrix
	MOV BL, [DI]	Load BL with data of second matrix
	ADD AL, BL	Add two data of AL and BL
	MOV [DI], AL	Store AL with data into DI
	INC DI	Increment DI
	INC SI	Increment SI
	DEC CL	Decrement CL
	JNZ NEXT	Loop continues until all elements of Matrix to added
	HLT	Halt the Program

Example:**Input:****Manual Calculation:****Matrix A**

2000: 00H
 2001: 01H
 2002: 02H
 2003: 03H
 2004: 04H
 2005: 05H
 2006: 06H
 2007: 07H
 2008: 08H

Matrix B

3000: 09H
3001: 08H
3002: 07H
3003: 06H
3004: 05H
3005: 04H
3006: 03H
3007: 02H
3008: 01H

Output

3000: 09H
3001: 09H
3002: 09H
3003: 09H
3004: 09H
3005: 09H
3006: 09H
3007: 09H
3008: 09H

REVIEW QUESTIONS:

1. Write the function of the following 8085 instructions: JP, JPE, JPO, and JNZ.
2. What is the purpose of the following commands in 8086?
 - a) AAD
 - b) RCL
3. List out the addressing modes in 8086.
4. What are the 8086 instructions used for BCD arithmetic?
5. What flags get affected after executing ADD instruction?

Result:

Thus the program for Matrix addition was successfully executed.

MATRIX MULTIPLICATION

Description:

The matrix multiplication is performed by loading the number of rows in CH reg and number of columns in CL reg and then multiplying the individual elements of the matrix.

Algorithm:

1. Initialize CH reg with no of rows
2. Initialize BX reg to 1400H
3. Initialize SI to 1200H
4. Initialize DI to 1300
5. Initialize CL reg with no of columns
6. Move 03 to DL
7. Initialize BP to 0000H
8. Initialize AX to 0000H
9. Store AH register into flags
10. Move the value pointed by SI to AL
11. Multiply the value pointed by DI with AL
12. Add the result with BP reg
13. Increment SI
14. Add 03 to point the next row element
15. Decrement DL
16. If not zero go to NEXT
17. Subtract DI with 08H
18. Subtract SI with 03H
19. Move the result to memory pointed by BP
20. Add 02 to BX
21. Decrement the value of CL
22. If not zero jump to COLUMN
23. Add 03H to SI
24. Decrement CH
25. If not Zero Jump to ROW
26. Halt

PROGRAM:

Label	Program	Comments
	MOV CH,03H	Initialize CH reg with no of rows
	MOV BX,1400H	Initialize BX reg to 1400H
	MOV SI,0200H	Initialize SI to 1200H
ROW:	MOV DI,1300H	Initialize DI to 1300
	MOV CL,03H	Initialize CL reg with no of columns
COLUMN	MOV DL,03H	Move 03 to DL
:	MOV BP,0000H	Initialize BP to 0000H
	MOV AX,0000H	Initialize AX to 0000H
	SAHF	Store AH register into flags

NEXT:	MOV AL,[SI]	Move the value pointed by SI to AL
	MUL [DI]	Multiply the value pointed by DI with AL
	ADD BP,AX	Add the result with BP reg
	INC SI	Increment SI
	ADD DI,03H	Add 03 to point the next row element
	DEC DL	Decrement DL
	JNZ NEXT	If not zero go to NEXT
	SUB DI,08H	Subtract DI with 08H
	SUB SI,03H	Subtract SI with 03H
	MOV [BX],BP	Move the result to memory pointed by BP
	ADD BX,02H	Add 02 to BX
	DEC CL	Decrement the value of CL
	JNZ COLUMN	If not zero jump to COLUMN
	ADD SI,03H	Add 03H to SI
	DEC CH	Decrement CH
	JNZ ROW	If not Zero Jump to ROW
	HLT	Halt

Example:

Input:

Matrix A

1200:02H
1201:02H
1202:02H
1203:02H
1204:02H
1205:02H
1206:02H
1207:02H
1208:02H

Matrix B

1300:02H
1301:02H
1302:02H
1303:02H
1304:02H
1305:02H
1306:02H
1307:02H
1308:02H

Output

1400:0CH
1401:00H

1402:0CH

1403:00H

1404:0CH

1405:00H

1406:0CH

1407:00H

1408:0CH

Manual Calculation:

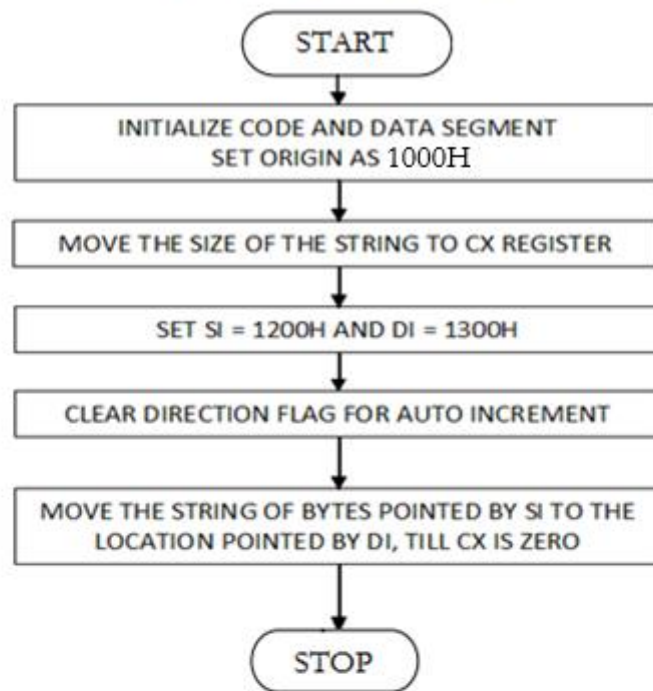
REVIEW QUESTIONS:

1. Write an ALP for 8086 to multiply two 16 bit unsigned numbers.
2. What is an accumulator?
3. Explain the uses of PUSH and POP instruction
4. When the 8086 processor is in minimum mode and maximum mode?
5. What is program counter?

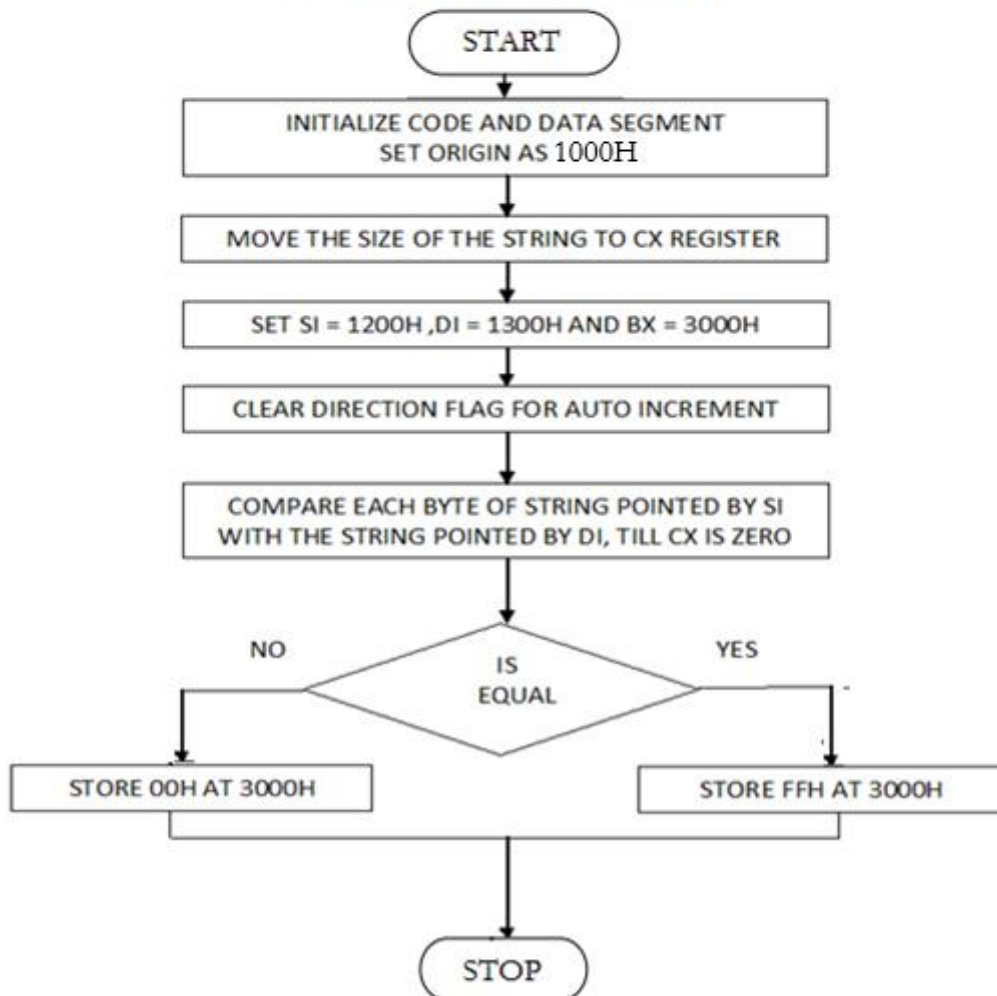
Result:

Thus the program for Matrix multiplication was successfully executed.

Flowchart for Copying a String



Flowchart for Comparing two Strings



Ex. No. 4

Date:

STRING MANIPULATION, SORTING AND SEARCHING

Objective:

To write an 8086 ALP to perform the following functions

- a) String Manipulation
 - Copying a String
 - Comparing Two Strings
 - Scan a character in a string
- b) Sorting
 - Ascending order
 - Descending order
- c) Searching

STRING MANIPULATION – COPYING A STRING

Description:

In 8086, a dedicated string instruction MOVSB is used to copy a string. On the MOVSB will move or copy the string of data pointed by SI to the location pointed by DI register on copying each byte of data, the SI register and DI register are incremented or decremented depending on the status of the direction flag DF. The CX register will hold the size of the string to be moved from one location to another location.

Algorithm:

1. Start the program.
2. Set the starting address as 1000H.
3. Get the array size & move it to CX segment.
4. Let the starting address of elements be 1200H & move it to SI.
5. Let starting address of another set of elements 1300H & move it to DI.
6. Clear Directional Flag.
7. Repeat the move single byte instruction till the count CX is zero.
8. End of segment.
9. Stop the program.

PROGRAM

Label	Program	Comments
	ORG 1000H	Set starting address as 1000H.
	MOV CX, 0005H	Move immediate data to CX.
	MOV SI, 1200H	Move immediate data to SI.
	MOV DI, 1300H	Move immediate data to DI.
	CLD	Clear Directional Flag.
	REP MOVSB	Repeat, Move single byte
	HLT	

Example:**Manual Calculation:****Input:**

1200: AA
 1201: AB
 1202: AC
 1203: DA
 1204: OA

Output:

1300: AA
 1301: AB
 1302: AC
 1303: DA
 1304: OA

STRING MANIPULATION – COMPARE TWO STRINGS**Description:**

In 8086, a dedicated string instruction CMPSB is used to compare two strings. The CMPSB will compare two strings of data pointed by SI and DI register. The REPE is used to repeat compare operation for each byte of the string. If both the strings are equal the CMPSB will set zero flag. If they are unequal ZF=0. The CX register will hold the size of the string.

In this program, if both the strings are equal, 00FFH is stored at 5000H else 0000H will be stored at 5000H.

Algorithm:

1. Start the program.
2. Set the starting address as 1000H.
3. Get array size and move it to CX register.
4. The starting address of a string is moved to SI register.
5. The starting address of another string is moved to DI register.
6. The BX register is initialized to point 3000H.
7. Clear directional flag
8. Compare each byte of string pointed by SI with the string pointed by DI till CX is zero.
9. If both the strings are equal, 0FFH is stored at the location pointed by BX register (3000H). Else store 00H at the location pointed by BX register.
10. End of the segment
11. Terminate the program

PROGRAM:

Label	Program	Comments
	ORG 1000H	Set starting address as 1000H.
	MOV CX, 0005H	Move immediate data to CX.
	MOV SI, 1200H	Move immediate data to SI.
	MOV DI, 1300H	Move immediate data to DI.

	MOV BX, 3000H	Move immediate data to BX.
	CLD	Clear directional flag.
	REPE CMPSB	Repeat if equal, compare single byte
	JNZ L1	Jump if no zero to loop1.
	MOV AH, 0FFH	Move immediate data to AH.
	MOV [BX], AH	Move AH to BX register
	JMP LAST	Jump to last.
L1:	MOV AH, 00H	Move immediate data to AH.
	MOV [BX], AH	Move AH to BX register.
LAST:	HLT	

Example:

Same String

Input:

1200: 02
1201: 03
1202: 04
1203: 05
1204: 06

1300: 02
1301: 03
1302: 04
1303: 05
1304: 06

Output:

3000: FFH

Different String

Input:

1200: 02
1201: 03
1202: 04
1203: 05
1204: 06

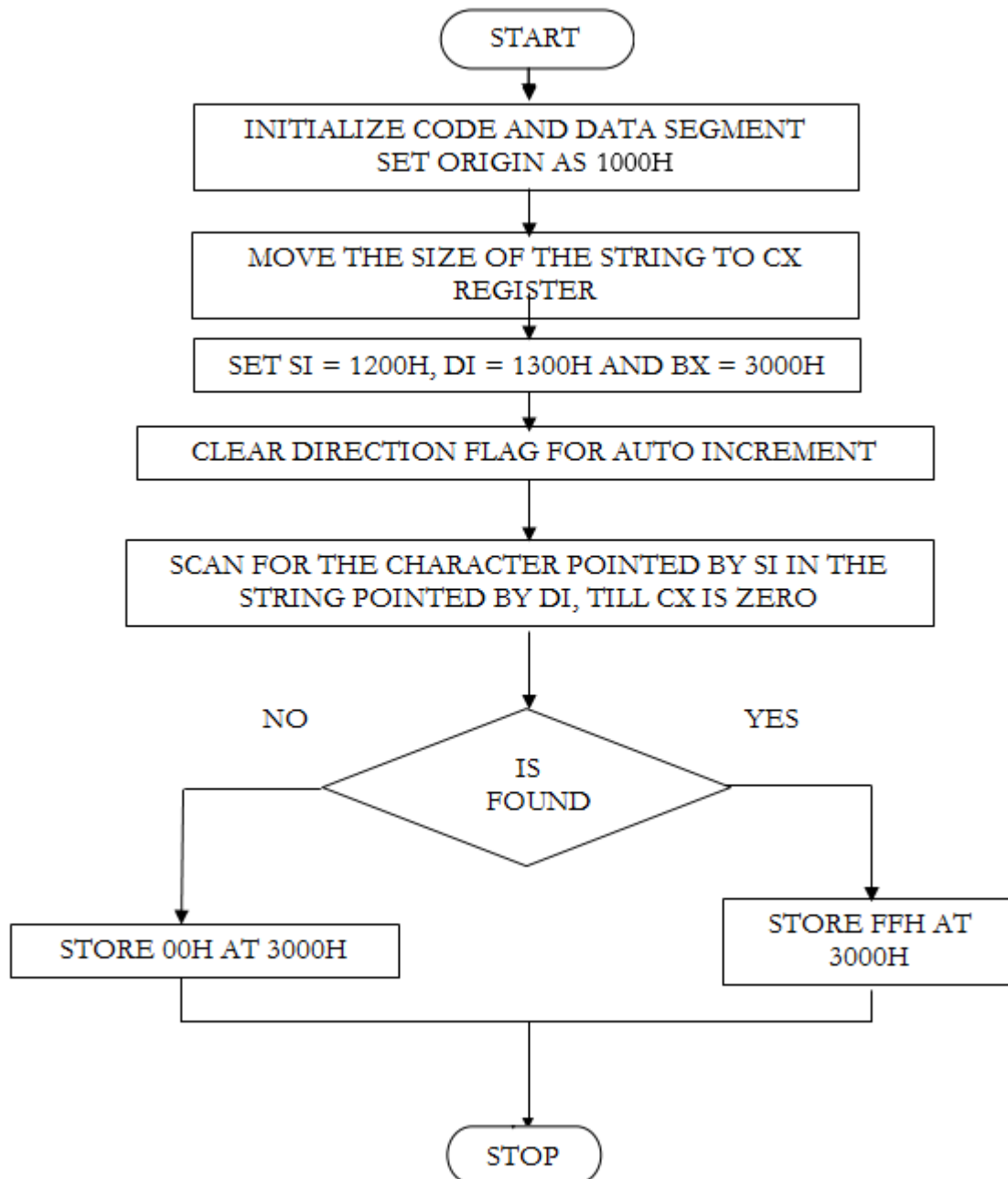
1300: 03
1301: 04
1302: 05
1303: 06
1304: 07

Output:

3000: 00H

Manual Calculation:

Flow Chart for Scan a Character in a String:



STRING MANIPULATION - SCAN A CHARACTER IN A STRING

Description:

In 8086, a dedicated string instruction SCASB is used to scan a character. The SCASB will scan for the character pointed by SI, in the string pointed by DI register. If the character is available in the string zero flag is set. Else zero flag is reset. The CX register will hold the size of the string.

In this program, if the given character is available 0FFH is stored at 5000H. If it is unavailable, 00H is stored at 5000H.

Algorithm:

1. Start the program.
2. Set the origin as 1000H.

3. Move the data pointed by SI to AL register.
4. Assign 0004H [count] to CX register.
5. The starting address of the string is moved to DI register
6. Clear Directional Flag for auto increment mode.
7. Repeatedly scan for the character at AL with DI till CX is zero.
8. If the character is found in the string, store 0FFH at location 3000H pointed by BX register. Else store 00H at location 3000H pointed by BX register.
9. End of segment.
10. Stop the program.

PROGRAM:

Label	Program	Comments
	ORG 1000H	Set the starting address as 1000H.
	MOV CX, 0004H	Move immediate data to CX.
	MOV SI, 1200H	Move immediate data to SI.
	MOV AL, [SI]	Move contents of SI to AL.
	MOV DI, 1300H	Move immediate data to DI.
	MOV BX, 3000H	Move immediate data to BX.
	CLD	Clear directional flag.
	REPNE SCASB	Repeat not equal, Scan single byte
	JNZ L1	Jump if no zero to loop1.
	MOV AH, 0FFH	Move immediate data to AH.
	JMP L2	Jump to loop 2.
L1:	MOV AH, 00H	Move immediate data to AH.
L2:	MOV [BX], AH	Move AH to BX register.
	HLT	

Example:

Input:

1200:AD (Data to be scanned)

1300:AA

1301:AB

1302:AA

1303:AD

Output:

3000:FF

Input:

1200: BB (Data to be scanned)

1300:AA

1301:AB

1302:AA

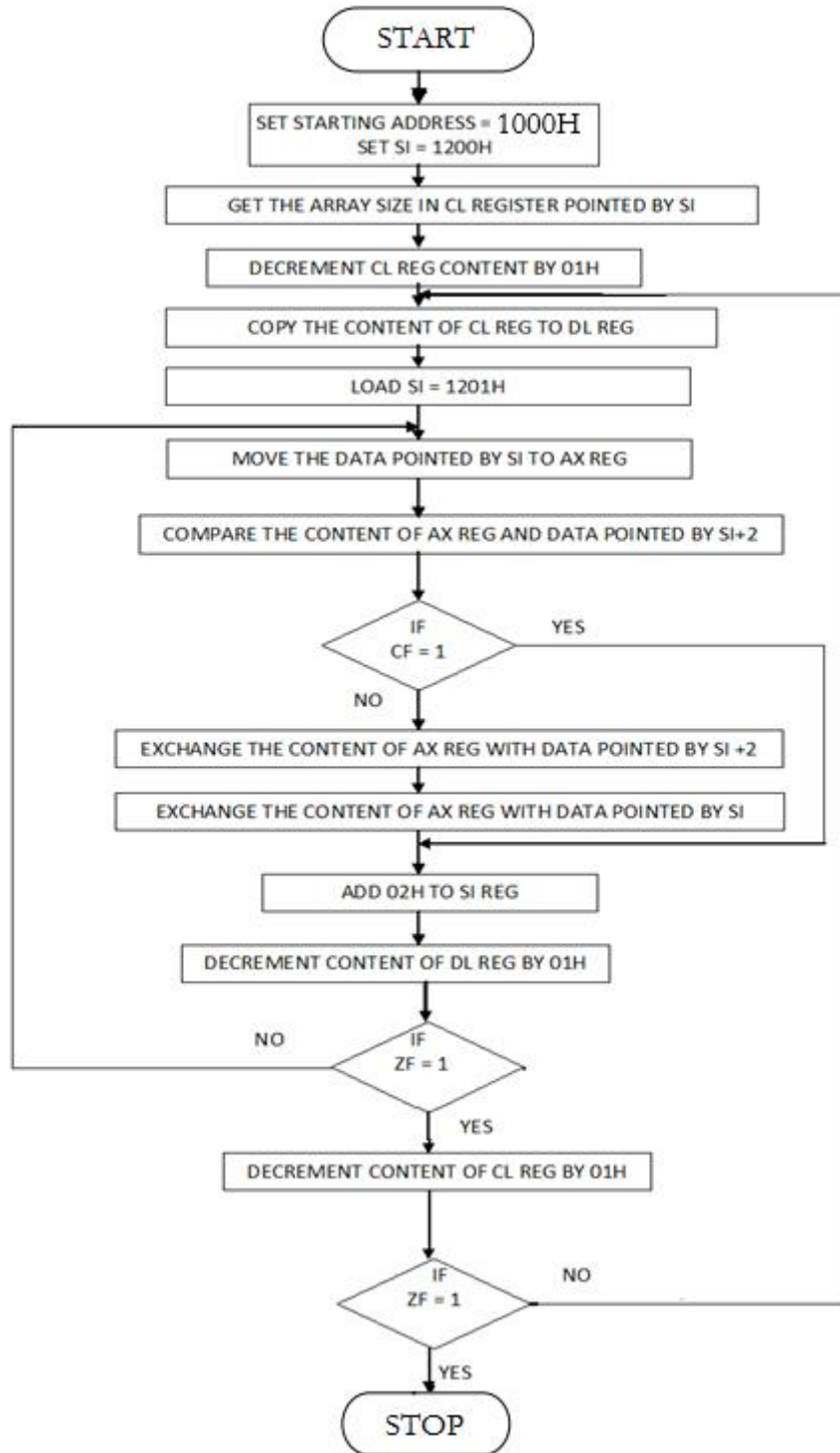
1303:AD

Output:

3000:00

Manual Calculation:

Flow Chart for Sorting – Ascending Order



SORTING – ASCENDING ORDER

Description:

The array can be sorted in ascending order by bubble sort algorithm. In bubble sorting of M-data, M-1 comparisons are performed by tasking two consecutive data at a time. After each comparison the two data can be re-arranged in the ascending order in the same memory locations i.e., smaller first and larger next. When the above M-1 comparisons are performed M-1 times, the array will be sorted in ascending order in the same locations.

Algorithm:

1. Start the program
2. Initialize Code and Data Segment.

3. Set starting address as 1000H
4. Set SI register to 1200H address
5. Get the count in CL & decrement CL register by one
6. Copy the content of CL register to DL register.
7. Initialize SI as 1202H.
8. Move the data pointed by SI to AX
9. Compare the data in AX & data pointed by SI+2
10. If there is no carry, exchange the data and go to next step. If there is carry go to next step.
11. Increment the content of SI by 02H
12. Decrement the content of DL register by 01H.
13. Check whether the content of DL is zero. If zero, go to step next step. Else go to step 8
14. Decrement the content of CL register by 01H.
15. Check whether the content of CL is zero. If zero, go to step next step. Else go to step 6
16. Display the result
17. Stop the program

PROGRAM:

Label	Program	Comments
	ORG 1000H MOV SI, 1200H MOV CL, [SI] DEC CL	Set starting address as 1000H. Move immediate data to SI Move contents of SI to CL Decrement CL
L3:	MOV DL, CL MOV SI, 1201H	Move CL to DL register Move immediate data to SI
L2:	MOV AX, [SI] CMP AX, [SI+2] JC L1 XCHG [SI+2], AX XCHG [SI], AX	Move contents of SI to AX Compare AX with SI Jump if carry to loop1 Exchange data of AX with SI+2 Exchange data of AX with SI
L1:	ADD SI, 02H DEC DL JNZ L2 DEC CL JNZ L3 HLT	Increment SI twice Decrement DL register Jump if no zero to loop 2 Decrement CL register Jump if no zero to loop 3

Example:

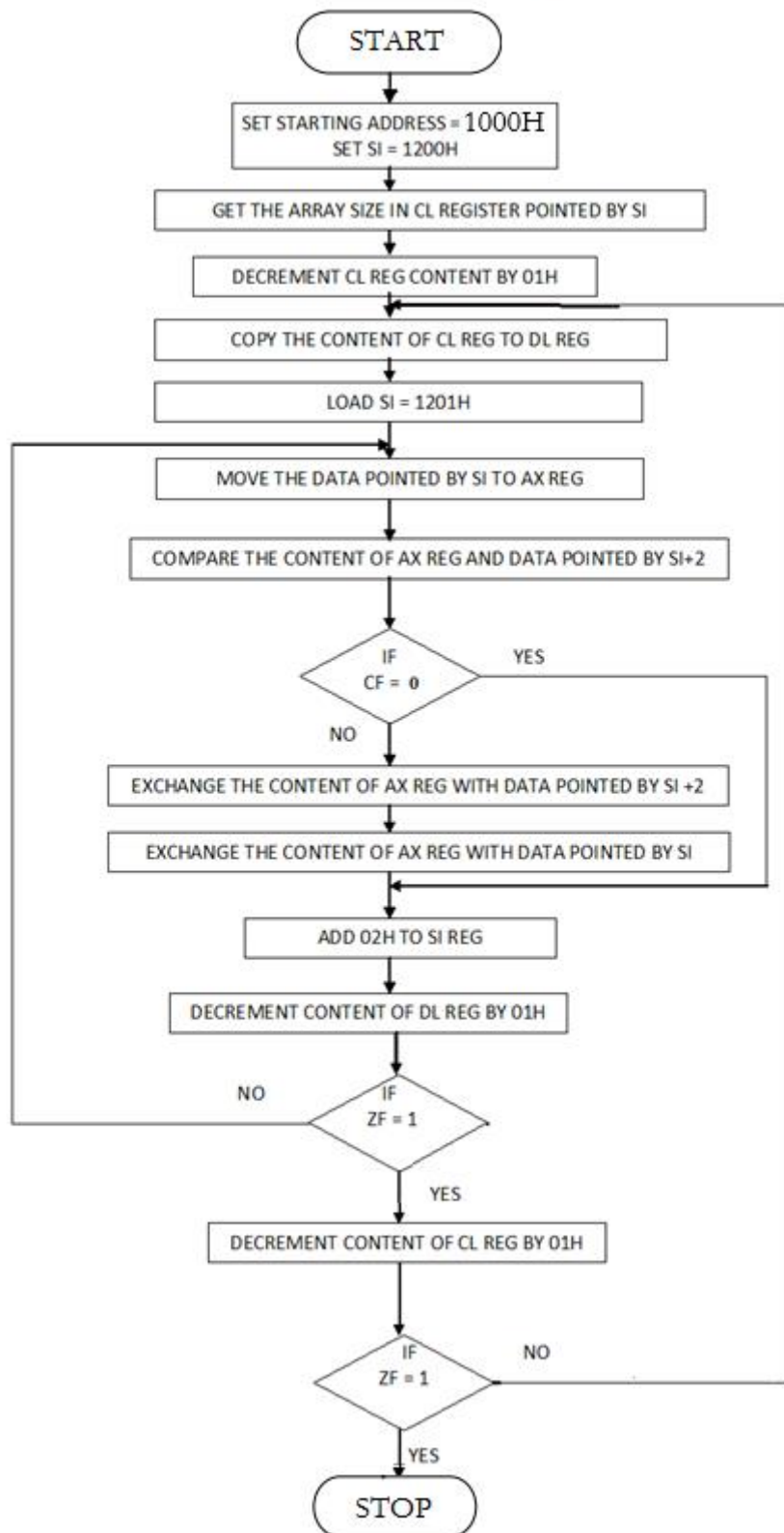
Input:

1200: 04 (Array Size)
1201: 39
1202: 40
1203: 30
1204: 78
1205: 62
1206: 42
1207: 32
1208: 38

Output:

1200: 04 (Array Size)
1201: 30
1202: 32
1203: 38
1204: 39
1205: 40
1206: 42
1207: 62
1208: 78

Flow Chart for Sorting - Descending Order



SORTING – DESCENDING ORDER

Description:

The array can be sorted in descending order by bubble sort algorithm. In bubble sorting of M-data, M-1 comparisons are performed by taking two consecutive data at a time. After each comparison, the two data can be re-arranged in the descending order in the same memory

locations, ie., larger first and smaller next. When the above M-1 comparisons are performed M-1 times, the array will be stored in descending order.

Algorithm:

1. Start the program
2. Set starting address as 1000H
3. Set SI register to 1200H address
4. Get the count in CL & decrement CL register by one
5. Copy the content of CL register to DL register.
6. Initialize SI as 1202H.
7. Move the data pointed by SI to AX
8. Compare the data in AX & data pointed by SI+2
9. If there is carry, exchange the data and go to next step. If there is no carry go to next step.
10. Increment the content of SI by 02H
11. Decrement the content of DL register by 01H.
12. Check whether the content of DL is zero. If zero, go to step next step. Else go to step 8
13. Decrement the content of CL register by 01H.
14. Check whether the content of CL is zero. If zero, go to step next step. Else go to step 6
15. Display the result
16. Stop the program

PROGRAM:

Label	Program	Comments
	ORG 1000H MOV SI, 1200H MOV CL, [SI] DEC CL	Set starting address as 1000H. Move immediate data to SI Move contents of SI to CL Decrement CL
L3:	MOV DL, CL MOV SI, 1201H	Move CL to DL register Move immediate data to SI
L2:	MOV AX, [SI] CMP AX, [SI+2] JNC L1 XCHG [SI+2], AX XCHG [SI], AX	Move contents of SI to AX register Compare SI+2 with AX register Jump if no carry to loop1 Exchange content of AX with SI+2 Exchange content of AX with SI
L1:	ADD SI, 02 DEC DL JNZ L2 DEC CL JNZ L3 HLT	Increment address of SI by 02 Decrement DL register Jump if no zero to loop 2 Decrement CL register Jump if no zero to loop 3

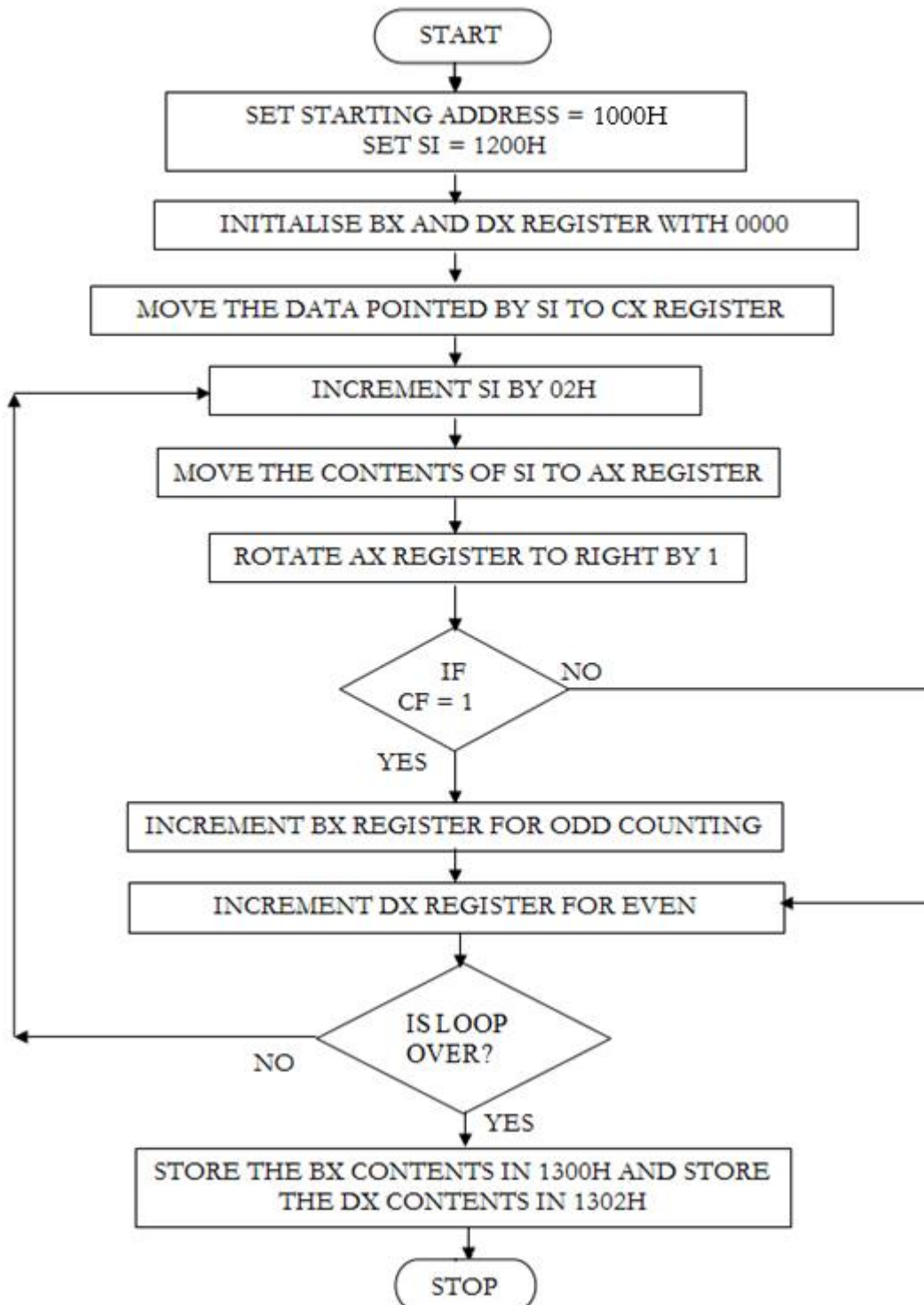
Example:

Input:	1204:78
	1205:62
1200: 04 (Array Size)	1206:42
1201:39	1207:32
1202:40	1208:38
1203:30	

Output:	1204:40
	1205:39
1200: 04 (Array Size)	1206:38
1201:78	1207:32
1202:62	1208:30
1203:42	

Manual Calculation:

Flow Chart for Searching Odd-Even Numbers:



SEARCHING – EVEN AND ODD NUMBERS

Description:

This program is used to count the number of even numbers and odd numbers in given array. Here one right rotate operation is performed to detect the even or odd number. After rotating operation, if carry is present, the given number is odd else it is even.

Algorithm:

1. Start the program
2. Initialize Code and Data Segment.
3. Set starting address as 1000H
4. Set SI register to 1200H address
5. Get the count in CL & decrement CL register by one
6. Initialize SI as 1202H.
7. Move the data pointed by SI to AX
8. Rotate AX register by right to one
9. If there is no carry, count the DX register for even counting else count the BX register for odd counting
10. Check loop is over or not
11. Increment the content of SI by 02H goto step 7.
12. Store the BX contents in 1300h
13. Store the DX contents in 1302h
14. Display the result
15. Stop the program

PROGRAM:

Label	Program	Comments
	ORG 1100H MOV SI, 1200H MOV DX, [SI] MOV CL, 01H MOV BL, 00H MOV BH, 00H	Set starting address as 1100H. Move immediate data to SI Move contents of SI to DX
L3:	ADD SI, 02H MOV AX, [SI] RCR AX, CLH JNC L1 INC BL JMP L2	INCREMENT SI BY 02H Move contents of SI to AX Rotate AX to right by one. Jump if no carry to loop1 count the BL register for odd counting Jump to l2
L1: L2:	INC BH DEC DX JNZ L3 MOV DI, 1300H MOV [DI], BL INC DI MOV [DI], BH HLT	count the BH register for even counting Count is performed until DX=0. Store the BL(ODD) contents in 1300h Store the BH(EVEN) contents in 1301h

Example:**Input:**

1200:05 (Array Size)
1201:00
1202:01
1203:02
1204:04
1205:06

Manual Calculation:**Output:**

1300:01 odd
1301:03 even

REVIEW QUESTIONS:

1. What is the relation between 8086 processor frequency & crystal Frequency?
2. What is the position of the stack pointer after the POP instruction?
3. Can ROM be used as stack?
4. Define – Baud Rate
5. What is cache memory?

Result:

Thus the program for string manipulations, searching and sorting operations was written and executed.

Ex. No. 5

Date:

PASSWORD CHECKING, PRINT RAM SIZE, SYSTEM DATE

Objective:

To write an 8086 ALP to perform the following operations

- d) Password Checking
- e) Print RAM Size
- f) Print System Date

PASSWORD CHECKING

Description:

The password checking is done using the DOS calls and functions. First Display the message "Enter your Password". Then read the pass word using Dos calls and compare with previous password "MASM1234". If it matches, then display the message password is correct. Else display it as incorrect password

Algorithm:

1. Start the program.
2. Set the starting address as 1000H.
3. Display the message "Enter your Password"
4. Read the pass word using Dos calls and compare with previous password "MASM1234"
5. If it matches, then display the message password is correct
6. Else display it as incorrect password
7. Stop the program.

PROGRAM:

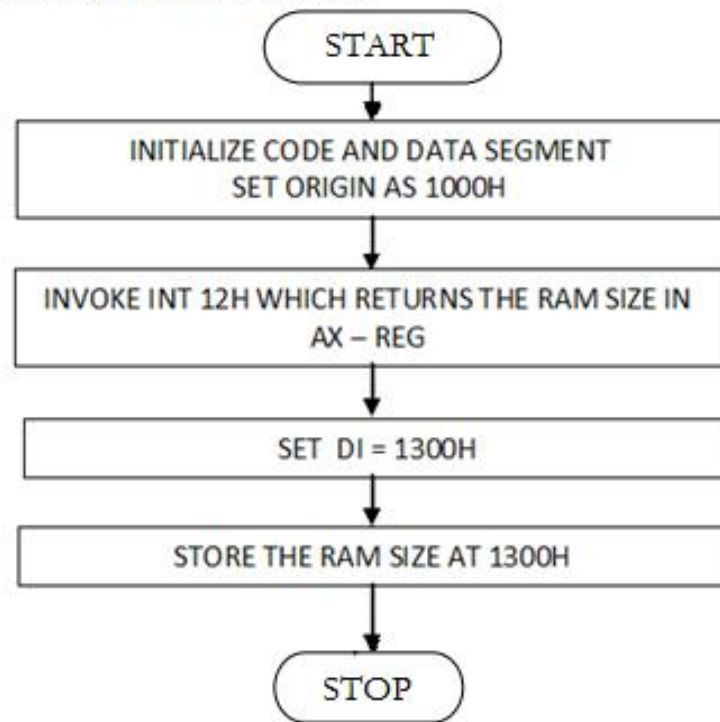
Label	Program	Comments
	DATA SEGMENT PASSWORD DB 'MASM1234' LEN EQU (\$-PASSWORD) MSG1 DB 10,13,'ENTER YOUR PASSWORD: \$' MSG2 DB 10,13,'YOUR PASSWORD IS CORRECT!!\$' MSG3 DB 10,13,'INCORRECT PASSWORD!\$' NEW DB 10,13,'\$' INST DB 10 DUP(0) DATA ENDS CODE SEGMENT ASSUME CS:CODE,DS:DATA ORG 1000H START: MOV AX,DATA	

	MOV DS,AX LEA DX,MSG1 MOV AH,09H INT 21H MOV SI,00 UP1: MOV AH,08H INT 21H CMP AL,0DH JE DOWN MOV [INST+SI],AL	
--	--	--

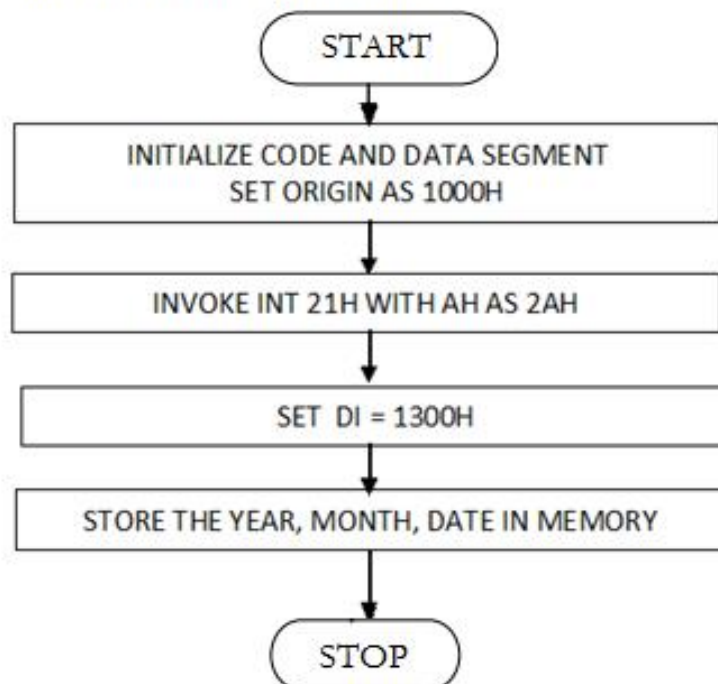
Label	Program	Comments
	MOV [INST+SI],AL MOV DL,'*' MOV AH,02H INT 21H INC SI JMP UP1 DOWN: MOV BX,00 MOV CX,LEN CHECK: MOV AL,[INST+BX] MOV DL,[PASSWORD+BX] CMP AL,DL JNE FAIL INC BX LOOP CHECK LEA DX,MSG2 MOV AH,09H INT 21H JMP FINISH FAIL: LEA DX,MSG3 MOV AH,009H INT 21H FINISH: INT 3 CODE ENDS END START END	

Observation:

Flow Chart to Print RAM Size



Flow Chart to Print System Date



TO PRINT RAM SIZE

Description:

INT 12h interrupt stores in **AX** the amount of RAM memory in kilobytes. For modern computers it usually returns the value 0280h (640), representing the main memory. So this interrupt doesn't return the extended memory. The value returned in **AX** by this interrupt could also be found at address 0040:0013h.

Algorithm:

1. Start the program.
2. Initialize the Segments.
3. Set the starting address as 1000H.
4. Initiate INT21H which returns the RAM size in AX – reg.
5. Initialize DI as 1300H
6. Store the value at 1300H
7. End of the segment
8. Terminate the program

PROGRAM:

Label	Program	Comments
	ASSUME CS:CODE,DS:CODE CODE SEGMENT ORG 1000H INT 12H MOV DI, 1300H MOV [DI],AX MOV AH,4CH INT 21H CODE ENDS	Initialize Segments Set the starting address as 1000H 12H interrupt is invoked Store the size of the RAM at 1300H

Example:**Output:**

1300: 80

Manual Calculation:**Program:**

Label	Program	Comments
	ASSUME CS:CODE,DS:CODE CODE SEGMENT ORG 1000H MOV AH,2AH INT 21H MOV DI, 1300H MOV [DI],CX ADD DI,02H MOV [DI],DX MOV AH,4CH INT 21H	Initialize Segments Set the starting address as 1000H 21H interrupt is invoked Store the year at 1300H Store the value of Month and day

	CODE ENDS	
--	-----------	--

Manual Calculation:

TO PRINT SYSTEM DATE

Description:

INT 21h interrupt with AH as 2AH will return the system date. The year (1980 – 2099) will be returned in CX register. The month will be available in DH register and day will be available in DL register. All the returned values will be in Hex.

Algorithm:

1. Start the program.
2. Initialize the Segments.
3. Set the starting address as 1000H.
4. Initiate INT21H with AH value as 2A H.
5. Initialize DI as 1300H
6. Store the value of year at 1300H
7. Store the value of Month and Day in the consecutive memory locations
8. End of the segment
9. Terminate the program

Example:

Manual Calculation:

Output:

```

1300: D    (Year)
1301: 07
1302: 0B   (Day)
1303: 08   (Month)

```

REVIEW QUESTIONS:

1. What is the role of Stack?
2. What is the difference between DOS and BIOS interrupts?
3. What is an interrupt vector Tabulation: of 8086?
4. Define – Machine cycle and T-State.
5. Define – Interrupt Vector Tabulation

Result:

Thus the program for password checking, printing RAM size, and System date was written and executed.

CYCLE II
Peripherals &
Interfacing Programs
(8086)

Ex. No. 6

Date:

INTERFACING TRAFFIC LIGHT CONTROL

AIM

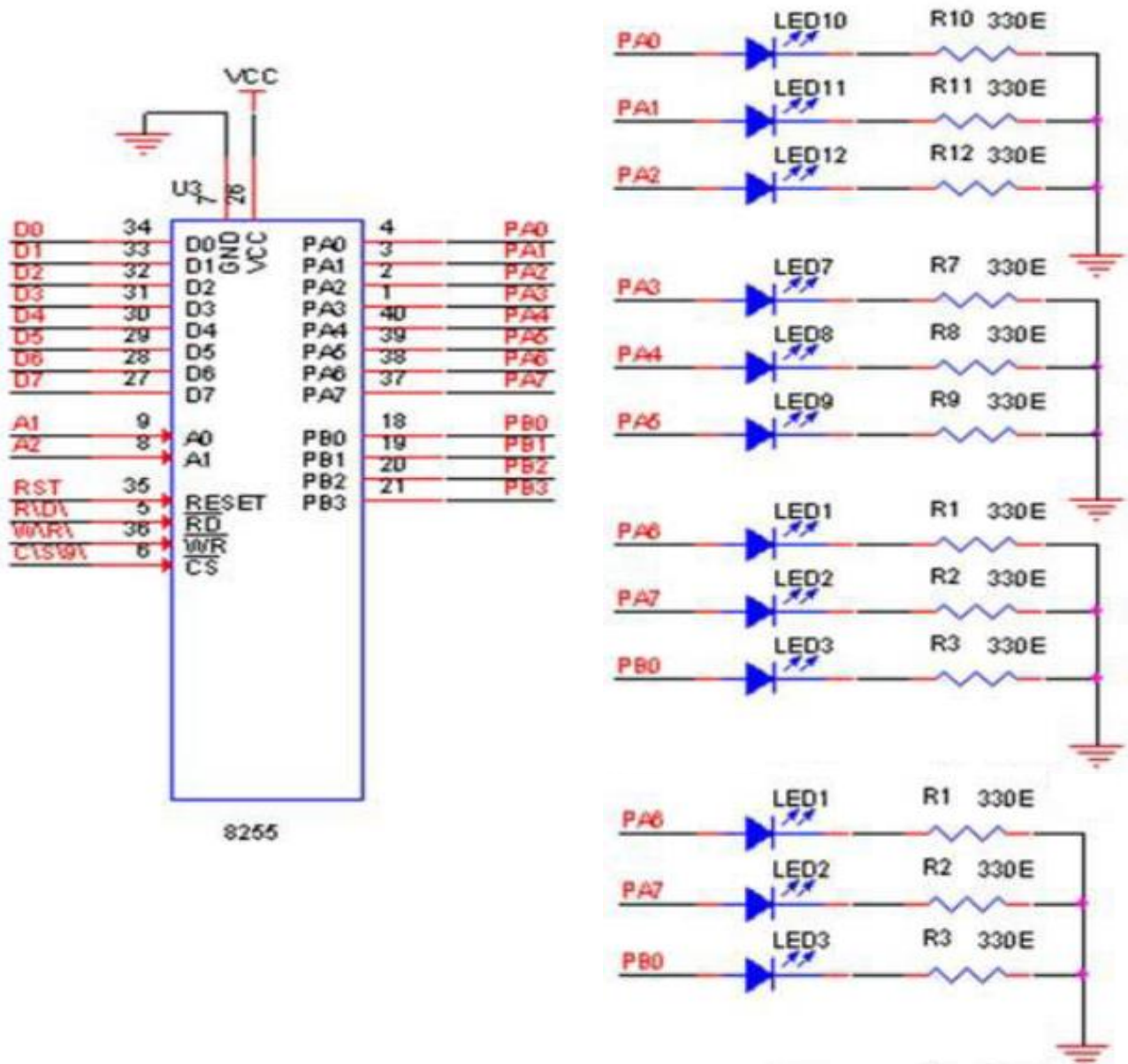
To write an 8086 assembly language program to interface the traffic light controller with 8255 and verify the operation.

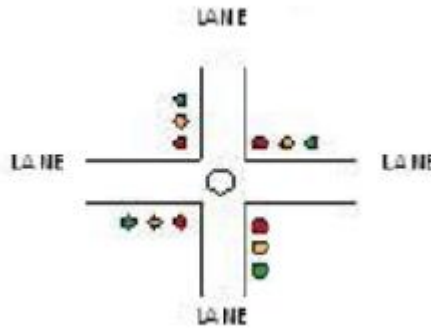
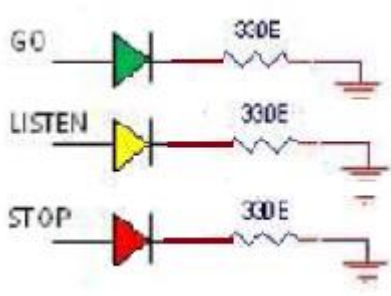
DESCRIPTION

The system is a simple contraction of a traffic control system wherein the signaling lights are simulated by the blinking or ON-OFF control of light-emitting diodes. The signaling lights for the pedestrian crossing are simulated by the ON-OFF control of dual colour light emitting diodes.

A model of a four road – four lane junctions, the board has green, orange and red signals of an actual system. Twelve LEDs are used on the board. In addition eight dual colour LEDs are used which can be made to change either to red or to green.

CIRCUIT DIAGRAM TO INTERFACE TRAFFIC LIGHT WITH 8086





Make high to - LED On

Make low to - LED Off

PROGRAM:

Label	Mnemonics
ART	ORG 1100H MOV BX, 1200 MOV CX, 000C MOV AL, [BX] OUT 26, AL INC BX MOV AL, [BX] OUT 20, AL INC BX MOV AL, [BX] OUT 22, AL CALL DELAY INC BX LOOP NEXT JMP START
DELAY	PUSH CX MOV CX, 0005
REPEAT AGAIN	MOV DX, FFFF DEC DX JNZ AGAIN LOOP REPEAT POP CX RET

**OBSERVATION
INPUT**

OUTPUT

1200: 80, 1A, A1, 64
1204: A4, 81, 5A, 64
1208: 54, 8A, B1, A8
120C: B4, 88, DA, 68
1210: D8, 1A, E8, 46
1214: E8, 83, 78, 86, 74

REVIEW QUESTIONS:

1. List out the control ports in traffic light controller
2. What are the functions of conditional instructions?
3. List out the LAN ports in traffic light controller
4. What are the functions of Loop instructions?
5. List out the Modules in traffic light controller

RESULT

Thus the interface the traffic light controller using 8086 microprocessors with 8255 has been executed and verified.

Ex. No. :7

Date:

INTERFACING STEPPER MOTOR AND TEMPERATURE SENSOR

Aim:

To write C Programs for running stepper motor either in clock- wise or counter-clock- wise and the direction of the rotation of the stepper motor depends on the variation in the temperature sensor.

Pre Lab Questions

1. What is LM35?
2. List the devices used to sense temperature.
3. What is the purpose of a thermocouple?
4. What is signal conditioning?
5. What is the output voltage of a thermocouple?

Apparatus & Software Required

1. LPC2148 Development board.
2. Keil μ V isi on5 software.
3. Flash Magic.
4. USB cable.
5. Stepper Motor.

Theory:

Stepper motors, effectively have multiple "toothed" electromagnets arranged around a central metal gear. To make the motor shaft turn, first one electromagnet is given power, which makes the gear's teeth magnetically attracted to the electromagnet's teeth. When the gear's teeth are thus aligned to the first electromagnet, they are slightly offset from the next electromagnet. So when the next electromagnet is turned on and the first will turn off, the gear rotates slightly to align with the next one and from there the process is repeated. Each of those slight rotations is called a "step." In that way, the motor can be turned to a précised angle. There are two basic arrangements for the electromagnetic coils: bipolar and unipolar.

Procedure:

1. Follow the steps to create a New project
2. Type the below code and save it with the name (any name.c)
3. Follow the steps to create a New Project to compile and build the program
4. Follow the procedures in to download your Hex code to processor using Flash Magic Software.

STEPPER MOTOR PROGRAM

```
/* This is a test program to stepper motor interface in the ARM LPC2148*/
/* development board itself*/
/*****
*****/
#include <LPC214x.H>    /* LPC214x definitions */

#define step1          0x00010000 /* P1.16
*/ #define step2      0x00020000 /* P1.17
*/
```

```

void wait (void)
{
    /* wait function */
    int d;
    for (d = 0; d < 10000; d++); /* only to delay for LED flashes */
}

call_stepper_forw()
{
    IOCLR1 = 0X00FF0000;
    IOSET1 = 0X00040000;
// wait();
// wait();
    wait();
    wait();
    IOCLR1 = 0X00FF0000;
    IOSET1 = 0X00060000;
// wait();
// wait();

    wait();
    wait();
    IOCLR1 = 0X00FF0000;
    IOSET1 = 0X00070000;
// wait();
// wait();
    wait();
    wait();
    IOCLR1 = 0X00FF0000;
    IOSET1 = 0X00050000;
// wait();
// wait();
    wait();
    wait();
}

int main (void)
{
    IODIR1 |=
        0xFFFFFFFF;
    IOCLR1|=
        0X00FF0000;
    wait();
    while(1) /*Loop Forever*/
    {
        call_stepper_forw();
        //wait();
        //wait();
        wait();
        wait();
        IOCLR1 = 0X00FF0000;
    }
}

```


STEPPER MOTOR PROGRAM PORT DETAILS

ARM	DETAILS
P1.16	STEP 1
P1.17	STEP 2

TEMPERATURE SENSOR PROGRAM

```

/*****/

MAIN ADC TEST

/*****/

/* This is a test program to temperature sensor in the ARM LPC2148 development board*/
/*****/
#include <LPC214x.H>          /* LPC214x definitions */
#include "ADC_Driver.c"      /* contains prototypes of driver functions*/
#include "lcd.c"
#include <stdio.h>

int main (void)
{
    unsigned    int
    adc_val;    unsigned
    int temp;
    unsigned char buf[4] = {0,0,0,0};
    ADCInit();
    lcdinit();
    //wait();
    clrscr(10
    );
    printstr("ADC
    Test",0,0); wait();
    while (1)                /* Loop forever */
    {
        adc_val = ADC_ReadChannel();
        temp = (unsigned int)((3*adc_val*100)/1024);
        sprintf(buf,"%d",temp);
        printstr(buf,0,1);

    }
}

/*****/

LCD.C

/*****/

*****/

#include <LPC214x.h>

#define RS      0x00000400 /* P0.10
*/ #define CE   0x00001800 /* P1.11
*/

```

```

void clrscr(char ch);
void lcdinit(void);
void lcdcmd(char);
void lcddat(char);
void gotoxy(char,char); //x,y ; x-char position(0 - 16) y-line number 0 or 1
void printstr(unsigned char *,char,char); //string,column(x),line(y)
void wait (void);
void split_numbers(unsigned int number);

```

```

#define SET 1
#define OFF 0

```

```

unsigned int thousands,hundreds,tens,ones;

```

```

void wait (void)    {                               /* wait function
    /*
    int d;
    for (d = 0; d < 100000; d++);                 /* only to delay for LED flashes */
}

```

```

void lcdinit()
{
    IODIR0 |= 0xFFFFFFFF;
    IOCLR0 |=0X00000FFF;
    lcdcmd(0x
    28);
    lcdcmd(0x28)
    ;
    lcdcmd(0x0c)
    ;
    lcdcmd(0x06)
    ;
    lcdcmd(0x01)
    ;
    lcdcmd(0x0f);
    wait();
}

```

```

void gotoxy(char x, char y)
{
    if(y == 0)
        lcdcmd(0x80+x);
    else
        lcdcmd(0xc0+x);
}

```

```

void printstr(unsigned char *str, char x, char y)
{
    char    i;
    gotoxy(x,y);

```

```

    wait();//(500)
    ;
    for(i=0;str[i]!='\0';i++)
        lcddat(str[i]);
}
void lcdcmd(char cmd)
{
    unsigned char LCDDAT;
        LCDDAT = (cmd & 0xf0);    //higher
        nibble IOSET0 = LCDDAT;
        IOCLR0 = RS; IOSET0 =
            CE;
        wait();//(100);           //enable lcd
        IOCLR0 = CE;
        IOCLR0 = 0X00000FFF;

        LCDDAT = ((cmd<<0x04) & 0xf0);    //lower
        nibble IOSET0 = LCDDAT;
        IOCLR0 = RS;
        IOSET0 = CE;
        wait();//(100);           //enable lcd
        IOCLR0 = CE;
        IOCLR0 = 0X00000FFF;
}

void lcddat(char cmd)
{
    unsigned char LCDDAT;
        LCDDAT = (cmd & 0xf0);    //higher
        nibble IOSET0 = LCDDAT;
        IOSET0 = RS;
        IOSET0 = CE;
        wait();//(100);           //enable lcd
        IOCLR0 = CE;
        IOCLR0 = 0X00000FFF;

        LCDDAT = ((cmd<<0x04) & 0xf0);    //lower
        nibble IOSET0 = LCDDAT;
        IOSET0 = RS;
        IOSET0 = CE;
        wait();//(100);           //enable lcd
        IOCLR0 = CE;
        IOCLR0 = 0X00000FFF;
}

void clrscr(char ch)
{
    if(ch==0)
    {
        printstr("",0,0);
        gotoxy(0,0);
    }
}

```

```

else if(ch == 1)
{
    printstr("",0,1);
    gotoxy(0,1);
}
else
{
    lcdcmd(0x01);
    // delay(100);
}
}

```

```

void split_numbers(unsigned int number)
{
    thousands = (number
/1000); number %= 1000;
    hundreds = (number /
100); number %= 100;
    tens = (number /
10); number %= 10;
    ones = number ;
}

```

```

void Wait_Msg(void)
{
    lcdcmd(0x01);
    printstr("Please Wait ", 0, 0);
}

```

```

void Welcome_Msg(void)
{
    lcdcmd(0x01);
    printstr("Welcome to      ", 0, 0);
    printstr("SMMICRRO      ", 0, 1);
}

```

ADC DRIVER.C

/*****

```

#include <LPC214x.H>          /* LPC214x definitions
*/
Void ADCInit (void)
{
    PINSEL1 |= 0x04000000; /*For Channel AD0.2 is
P0.29*/
    IODIR0 |= ~(0x04000000);
    AD0CR |= 0x00200204; /*0x04 selects AD0.2 to mux output, 0x20 makes ADC
in operational*/
    AD0GDR; /*A read on AD0GDR clears the DONE bit*/
}
void ADC_StartConversion(void)
{
    AD0CR |= (1<<24);
}
void ADC_StopConversion(void)

```

```

{
    AD0CR &= (~(1<<24));
}
unsigned int ADC_ReadChannel(void)
{
//    unsigned int i;
    unsigned long ADC_Val, t;
    ADC_StartConversion();

    while((AD0DR2&0x80000000)==0); /*wait until ADC conversion completes*/
    if(AD0STAT & 0x00000400)
        {
            //printstr("OVR",0,1)
            ; return(0);
        }
    t = AD0DR2;
    ADC_Val = ((t>>6) & 0x000003FF);/(AD0DR2 & 0x000003FF); (((AD0CR>>6) &
0x000003FF);
//ADC_StopConversion();
    return(ADC_Val);
}

```

TEMPERATURE SENSOR PROGRAM PORT DETAILS

ARM	DETAILS
P0.29	ADC0.2
PO.10	RS LCD PIN
P1.11	CE LCD PIN

REVIEW QUESTIONS:

1. Why LM35 is used to Measure Temperature?
2. Compare the difference between LM 34 and LM 35 sensors?
3. What is the operating temperature range in LM35?
4. How many pins are available in LM35?
5. What is the main function of analog pin in LPC 2148?

Result:

The C-Language program for running stepper motor either in clock-wise or counter-clock-wise Depending on the temperature is developed in the sensor LM35 and the output is verified in LCD.

Ex. No. : 8

Date:

Implementing zigbee protocol with ARM

Aim:

To write C Programs for Zigbee Protocol and verify the communication between Xbee Module Transmitter and Receiver.

Pre Lab Questions

1. What are the applications of zigbee protocol?
2. Why Zigbee based is preferred for wireless communication?
3. What is the function of a scheduler?
4. What is the main function of voltage convertors in UART?
5. List the advantages of using Zigbee protocol.

Apparatus & Software Required:

1. LPC2148 Development board.
2. Keil μ V isi on5 software.
3. Flash Magic.
4. USB cable.
5. Zigbee Mod ule Tx a nd Rx .

Theory:

The X Bee/X Bee-PRO ZNet 2.5 (formerly known as Series 2 and Series 2 PRO) RF Modules were directed to operate within the ZigBee protocol. The modules provide reliable delivery of data between remote devices. Zigbee is the communication protocol like wifi and Bluetooth. Xbee is the module using Zigbee protocol.

Some of its features are:

- ZigBee is targeted at radio-frequency (RF) applications
- Low data rate, long battery life, and secure networking
- Transmission range is between 10 and 75 meters (33~246 feet)
- The addressing space allows of extreme node density up to 18,450,000,000,000,000 devices (64 bit IEEE address)
- Using local addressing, simple networks of more than 65,000 nodes can be configured, with reduced address overhead
- The radios use direct-sequence spread spectrum coding, which is managed by the digital stream into the modulator.
- To ensure reliable data transmission
- Binary phase shift keying (BPSK) in the 868/915 MHz
- Offset quadrature phase shift keying (O-QPSK) at 2.4 GHz

Procedure:

1. Follow the steps to create a New project
2. Type the below code and save it with the name (anyname.c)
3. Follow the steps to create a New Project to compile and build the program
4. Follow the procedures in to download your Hex code to processor using Flash Magic Software.

```
/******
```

ARM TRANSMITTER
PROGRAM LCD.C

```
*****
```

```
#include
<LPC214x.h>
#include "lcd.h"
#define RS      0x00000400 /* P0.10
*/ #define CE   0x00001800 /* P1.11
*/

/*void clrscr(char ch);
void  lcdinit(void);
void lcdcmd(char);
void lcdat(char);
void gotoxy(char,char); //x,y ; x-char position(0 - 16) y-line number 0 or 1 void
printstr(char *,char,char); //string,column(x),line(y)
void wait (void);
void split_numbers(unsigned int number);*/

#define SET 1
#define OFF 0
unsigned int thousands,hundreds,tens,ones;
void wait (void)
{
    /* wait function */
    int  d;
    for (d = 0; d < 100000; d++); /* only to delay for LED flashes */
}

void lcdinit()
{
    IODIR0      |=
        0xFFFFFFFF;
    IOCLR0      |=
        0X00000FFF;
    lcdcmd(0x
28);
    lcdcmd(0x28)
;
    lcdcmd(0x0c)
;
    lcdcmd(0x06)
;
    lcdcmd(0x01)
;
    lcdcmd(0x0f);
    wait();
}

void gotoxy(char x, char y)
{
    if(y == 0)
```

```

        lcdcmd(0x80+x);
    else
        lcdcmd(0xc0+x);

}

void printstr(char *str, char x, char y)
{
    char    i;
    gotoxy(x,y);
    wait();//(500)
    ;
    for(i=0;str[i]!='\0';i++)
    lcdat(str[i]);

}

void lcdcmd(char cmd)
{
    unsigned char LCDDAT;
        LCDDAT = (cmd & 0xf0);    //higher
        nibble IOSET0 = LCDDAT;
        IOCLR0 = RS; IOSET0 =
        CE;
        wait();                    //(100);
                                    //enable lcd

        IOCLR0 = CE;
        IOCLR0 = 0X00000FFF;

        LCDDAT = ((cmd<<0x04) & 0xf0);    //lower
        nibble IOSET0 = LCDDAT;
        IOCLR0 = RS;
        IOSET0 = CE;
        wait();//(100);            //enable lcd
        IOCLR0 = CE;
        IOCLR0 = 0X00000FFF;

}

void lcdat(char cmd)
{
    unsigned char LCDDAT;
        LCDDAT = (cmd & 0xf0);    //higher
        nibble IOSET0 = LCDDAT;
        IOSET0 = RS; IOSET0 = CE;
        wait();//(100);            //enable lcd
        IOCLR0 = CE;
        IOCLR0 = 0X00000FFF;

        LCDDAT = ((cmd<<0x04) & 0xf0);    //lower
        nibble IOSET0 = LCDDAT;
        IOSET0 = RS; IOSET0 = CE;
        wait();//(100);            //enable lcd
        IOCLR0 = CE;
        IOCLR0 = 0X00000FFF;

}

```



```

void clrscr(char ch)
{
    if(ch==0)
    {
        printstr("",0,0);
        gotoxy(0,0);
    }
    else if(ch == 1)
    {
        printstr("",0,1);
        gotoxy(0,1);
    }
    else
    {
        lcdcmd(0x01);
        // delay(100);
    }
}

```

```

void split_numbers(unsigned int number)
{
    thousands = (number
/1000); number %= 1000;
    hundreds = (number /
100); number %= 100;
    tens = (number /
10); number %= 10;
    ones = number ;
}

```

```

void Wait_Msg(void)
{
    lcdcmd(0x01);
    printstr("Please Wait ", 0, 0);
}

```

```

void Welcome_Msg(void)
{
    lcdcmd(0x01);
    printstr("Welcome to      ", 0, 0);
    printstr("SM MICRRO      ", 0, 1);
}

```

```

/*****
*/

```

LCD.h

```

/*****
*****/

```

```

void clrscr(char ch);
void lcdinit(void);
void lcdcmd(char);
void lcdat(char);
void gotoxy(char,char); //x,y ; x-char position(0 - 16) y-line number 0 or 1
void printstr(char *,char,char); //string,column(x),line(y)

```



```

    for (d = 0; d < 100000; d++);          /* only to delay for LED flashes */
}

void tx_string(char str)
{
    putchar(str);
}

void putchar (unsigned char ch)           /* Writes character to Serial Port*/
{
    while (!(U1LSR & TEMT));              /* U1LSR --> Status register
*/
    U1THR = ch;
}

/*****
ARM RECEIVER PROGRAM
*****/

#include
<LPC214X.H>
#include "lcd.c"
void
uart_1(void);
void
delay(void);
unsigned char getcharr (void);           /* Reads character from Serial Port*/

int main(void)
{
char rx_data;
    uart_1(
    );
    lcdinit(
    );
    printstr("SM MICRRO SYSTEM",0,0);
    while(1)
    {

void uart_1(void)      /* UART Installation */
{
    PINSEL0 = 0x00050000;
    U1LCR = 0x83;
    U1FDR = 0x00000010;
    U1DLL = 98;
    U1LCR = 0x03;
    U1IER = 0x01;
}

void delay(void)
{
    int d;
    for (d = 0; d < 100000; d++);        /* only to delay for LED flashes */
}

unsigned char getcharr (void)           /* Reads character from Serial Port*/
{

```

```

while (!(U1LSR &
0x01));      return
(U1RBR);
}

```

Implementing zigbee protocol with ARM PROGRAMS PORT DETAIL

TRANSMITTER PROGRAM

RECEIVER PROGRAM

ARM	Details
P0.8	TXD1
P0.9	RXD1
P0.10	RS LCD PIN
P1.11	CE LCD PIN
ARM	Details
P0.8	TXD1
P0.9	RXD1
P0.10	RS LCD PIN
P1.11	CE LCD PIN

Post Lab Questions:

1. How to verify the communication between Transmitter and Receiver?
2. Which module is using Zigbee protocol?
3. How many UART ports available in LPC2148?
4. Write the two modes of communication are used in a ZigBee network.
5. Mention the transmission range for Zigbee protocol.

Result:

The C-Language program for Zigbee Protocol is written and the communication between Zigbee Module Transmitter and Receiver is verified.

Ex. No. : 9

Date:

INTERFACING KEYBOARD AND LCD MATRIX KEYBOARD PROGRAM

Aim:

To develop a C-Language program for displaying the Key pressed in the Keypad in the LCD module. The display should come in the desired line and column.

Pre Lab Questions

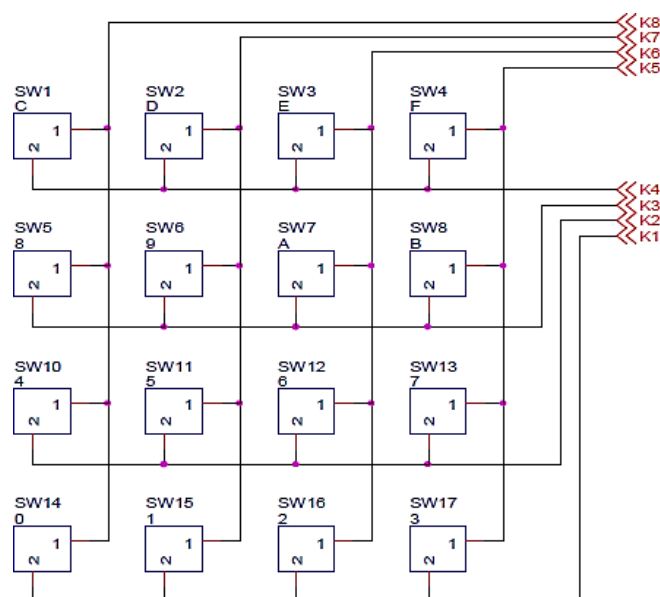
1. Mention the function of pull up resistor?
2. Outline the keyboard matrix.
3. Summarize the working principal of LCD.
4. What kind of interrupt is generated if a key has to be operated in an interrupt mode?
5. How many rows and columns are present in a 16 x 2 alphanumeric LCD?

Apparatus & Software Required

1. LPC2148 Development board.
2. Keil μ V isi on5 software.
3. Flash Magic.
4. USB cable.

Theory:

The Matrix keyboard is used to minimize the number of I/O lines. Normally it is possible to connect only one key or switch with an I/O line. If the number of keys in the system exceeds the more I/O lines are required. To reduce the number of I/O lines the keys are connected in the matrix circuit. Keyboards use a matrix with the rows and columns made up of wires. Each key acts like a switch. When a key is pressed a column wire makes contact with row wire and completes a circuit. For example, 16 keys arranged in a matrix circuit uses only 8 I/O lines.



Procedure:

1. Follow the steps to create a New project
2. Type the below code and save it with the name (anyname.c)
3. Follow the steps to create a New Project to compile and build the program
4. Follow the procedures in to download your Hex code to processor using Flash Magic Software.

```
/******  
  
                                MAIN.C  
/******  
*/  
/* Description: This program gets input from Matrix key board and displays  
corresponding */  
/*           Key value in 7segment display. Hence this program demonstrates both  
/*           7 segment display as well as Matrix key board. */  
/*           P1.16 to P1.23 are inputs from matrix key board,*/  
/*           P1.24 to P1.31 are outputs to 7 segment display  
*/  
/******  
**  
*****/  
/*           ----- matrix key board description-----  
*/  
/*           --           --           --           --  
*/  
/* row1  --| c |-- --| d |-- --| e |-- --| F |--           (SW1,SW2,SW3,SW4)  
*/  
/*           --           --           --           --  
*/  
/*           --           --           --           --  
*/  
/* row2  --| 8 |-- --| 9 |-- --| A |-- --| b |--           (SW5,SW6,SW7,SW8)  
*/  
/*           --           --           --           --  
*/  
/*           --           --           --           --  
*/  
/* row3  --| 4 |-- --| 5 |-- --| 6 |-- --| 7 |--           (SW9,SW10,SW11,SW12)  
*/  
/*           --           --           --           --  
*/  
/*           --           --           --           --  
*/  
/* row4  --| 0 |-- --| 1 |-- --| 2 |-- --| 3 |--           (SW13,SW14,SW15,SW16)  
*/  
/*           --           --           --           --*/  
/******  
#include  
<LPC214x.h>  
#include  
"mat_7seg.h"  
int main()  
{
```

```

unsigned int key, last_key, Disp_key;
init_Matrix_7seg(); // Initialize matrix keyboard and 7segment display
clearall_7seg(); // clear 7 segment display
last_key = 0; // Initialize this variable to zero
while(1)
{
    key = catch_key(); // scan for a valid key press
    if(key != 0) // zero means no key is pressed
    {
        if(key != last_key) // check whether the same key is pressed again(assume
this as STEP1)
        {
            Disp_key = key; // valid new key is stored in another variable
            last_key = key; // this variable's value is used for STEP1
        }
    }
    //Display_Number(Disp_key); /*this function is used to display number in
decimal format*/
    Alpha_Dispay(4,Disp_key); /*this function is used to display number in
hex format (single digit only)*/
}
}

```

```

/*****

```

MATRIX SEVEN SEGMENT DRIVER.C

```

*****/

```

```

#include<LPC214
x.h> #include
"defs.h"

```

```

/*****Global
variables*****/
unsigned int thousands,hundreds,tens,ones;

```

```

*****/
*****/

```

```

void init_Matrix_7seg(void)
{
    IODIR1 |= 0xff0f0000; // set 7seg LEDs as output ports and matrix's MSB as
inputs and LSB as outputs
    IODIR0 |= S7SEG_ENB; // set P0.19 to P0.22 as outputs to drive 7seg enable
pins

```

```

IOPIN0 |= S7SEG_ENB; // since we are using
active low 7 seg display, the
enable signals
// should be initially set to
HIGH.

```

```

/*****
**
*****/
unsigned long scan_row(unsigned int row_num)
{
    //unsigned int row,i;
    unsigned long val;

    IOSET1 = ROW_MASK;    //clear the previous scan row output ie
make all row ops high
    switch(row_num)
    {
        case 1: IOCLR1 = ROW1;break; // make P1.16
low
        case 2: IOCLR1 = ROW2;break; // make P1.17
low
        case 3: IOCLR1 = ROW3;break; // make P1.18
low
        case 4: IOCLR1 = ROW4;break; // make P1.19
low

        //default: row = ERR;
    }
//    for(i=0;i<=65000;i++);
    val = IOPIN1;    // read the matrix inputs
    val = ((val >> 20) & 0x0000000F)^0x0000000F; // shift the colum value so that it
comes to LSB
complement of shifted value. return(val);
}
// XORing is done to take 1's//

unsigned int catch_key(void)
{
    unsigned long
v;    v    =
scan_row(1);
switch(v)
{
    case 1: return(13);
    case 2: return(14);
    case 4: return(15);
    case 8: return(16);
}
v    =
scan_row(2
); switch(v)
{
    case 1: return(9);
    case 2: return(10);
    case 4: return(11);
    case 8: return(12);
}
v    =
scan_row(3
); switch(v)
{

```



```

*****/
void Digit_Dispay(int digit_num, unsigned int value)
{
    clearDigit_7seg(digit_num
); switch(value)
    {
        case 0: IOPIN1 |= ZERO;break;
        case 1: IOPIN1 |= ONE; break;
        case 2: IOPIN1 |= TWO; break;
        case 3: IOPIN1 |= THREE;
                break;
        case 4: IOPIN1 |= FOUR; break;
        case 5: IOPIN1 |= FIVE; break;
        case 6: IOPIN1 |= SIX; break;
        case 7: IOPIN1 |= SEVEN;
                break;
        case 8: IOPIN1 |= EIGHT; break;
        case 9: IOPIN1 |= NINE; break;
    }
}

/*****
**
*****/
void Alpha_Dispay(int digit_num, unsigned int value)
{
    clearDigit_7seg(digit_num
); switch(value)
    {
        case 1: IOPIN1 |= ZERO;break;
        case 2: IOPIN1 |= ONE; break;
        case 3: IOPIN1 |= TWO; break;
        case 4: IOPIN1 |= THREE;
                break;
        case 5: IOPIN1 |= FOUR; break;
        case 6: IOPIN1 |= FIVE; break;
        case 7: IOPIN1 |= SIX; break;
        case 8: IOPIN1 |= SEVEN;
                break;
        case 9: IOPIN1 |= EIGHT; break;
        case 10: IOPIN1 |= NINE;
                break; case 11: IOPIN1 |= AAA;
                break; case 12: IOPIN1 |= bbb;
                break; case 13: IOPIN1 |= ccc;
                break; case 14: IOPIN1 |= ddd;
                break; case 15: IOPIN1 |= eee;
                break; case 16: IOPIN1 |= fff;
                break;
    }
}

```

```
    }
    /**
    *****/
void split_numbers(unsigned int number)
{
    thousands = (number
/1000); number %= 1000;
    hundreds = (number /
100); number %= 100;
    tens = (number /
10); number %= 10;
```

```

    ones = number ;
}
/*****
**
*****/
void Display_Number(unsigned int num)
{
    unsigned int
    i; if(num <=
    9999)
    {
        clearall_7seg();
        split_numbers((unsigned
        int)num); Digit_Dispay(4, ones);
        for(i=0;i<10000;i++);
        Digit_Dispay(3,
        tens);
        for(i=0;i<10000;i++);
        ;
        Digit_Dispay(2,
        hundreds);
        for(i=0;i<10000;i++);
        Digit_Dispay(1,
        thousands);
        for(i=0;i<10000;i++);
    }
}
}

```

MATRIX SEVEN SEGMENT PROGRAM PORT DETAIL

ARM	DETAILS
P0.19	SEGMENT ENABLE PIN
P0.21	SEGMENT ENABLE PIN
P0.22	SEGMENT ENABLE PIN
P1.16	KEY BOARD INPUT
P1.17	KEY BOARD INPUT
P1.18	KEY BOARD INPUT
P1.19	KEY BOARD INPUT
P1.20	KEY BOARD INPUT
P1.21	KEY BOARD INPUT
P1.22	KEY BOARD INPUT
P1.23	KEY BOARD INPUT
P1.24	OUTPUT SEGMENT
P1.25	OUTPUT SEGMENT
P1.26	OUTPUT SEGMENT
P1.27	OUTPUT SEGMENT
P1.28	OUTPUT SEGMENT
P1.29	OUTPUT SEGMENT
P1.30	OUTPUT SEGMENT


```

;
lcdcmd(0x06)
;
lcdcmd(0x01)
;
lcdcmd(0x0f)
; wait();
}

void gotoxy(char x, char y)
{
    if(y == 0)
        lcdcmd(0x80+x);
    else
        lcdcmd(0xc0+x);
}

void printstr(char *str, char x, char y)
{
    char    i;
    gotoxy(x,y);
    wait();//(500)
    ;
    for(i=0;str[i]!='\0';i++)
        lcdat(str[i]);
}

void lcdcmd(char cmd)
{
    unsigned char LCDDAT;
    LCDDAT = (cmd & 0xf0);    //higher
    nibble IOSET0 = LCDDAT;
    IOCLR0   =
    RS;
    IOSET0 =
    CE;
    wait();//(100);           //enable lcd
    IOCLR0 = CE;
    IOCLR0 = 0X00000FFF;

    LCDDAT = ((cmd<<0x04) & 0xf0);    //lower
    nibble IOSET0 = LCDDAT;
    IOCLR0   =
    RS;
    IOSET0 =
    CE;
    wait();//(100);           //enable lcd
    IOCLR0 = CE;
    IOCLR0 = 0X00000FFF;
}

void lcdat(char cmd)

```

```

{
  unsigned char LCDDAT;
    LCDDAT = (cmd & 0xf0);    //higher
    nibble IOSET0 = LCDDAT;
  IOSET0   =
    RS;
    IOSET0 =
    CE;
  wait();//(100);              //enable lcd
    IOCLR0 = CE;
    IOCLR0 = 0X00000FFF;

    LCDDAT = ((cmd<<0x04) & 0xf0);    //lower
    nibble IOSET0 = LCDDAT;
  IOSET0   =
    RS;
    IOSET0 =
    CE;
  wait();//(100);              //enable lcd
    IOCLR0 = CE;
  IOCLR0 = 0X00000FFF;
}

```

```

void clrscr(char ch)
{
  if(ch==0)
  {
    printstr("",0,0);
    gotoxy(0,0);
  }
  else if(ch == 1)
  {
    printstr("",0,1);
    gotoxy(0,1);
  }
  else
  {
    lcdcmd(0x01);
    // delay(100);
  }
}

```

```

void split_numbers(unsigned int number)
{
  thousands = (number
/1000); number %= 1000;
  hundreds = (number /
100); number %= 100;
  tens = (number /
10); number %= 10;
  ones = number ;
}

```

```

void Wait_Msg(void)
{
    lcdcmd(0x01);
    printstr("    PLEASE WAIT    ", 0, 0);
}
void Welcome_Msg(void)
{
    lcdcmd(0x01);
    printstr("    WELCOME TO    ", 0, 0);
    printstr("SM MICRRO SYSTEM", 0, 1);
}

```

LCD main.c

```

/*****
*****/

/* This is a test program to display strings in LCD module in the ARM LPC2148 Development board itself
*/
/*****
*****/
#include <LPC214x.H>    /* LPC214x definitions */
#include "lcd.h"        /* includes lcd driver funtions*/

int main (void)
{
    lcdinit();          /*Initialize lcd*/
    Wait_Msg();         /*Display message - "Please Wait"*/
    Welcome_Msg();      /*Display message - "Welcome to SM
MICRRO"*/

    while(1)            /*Loop Forever*/
    {

    }
}

```

LCD PROGRAM PORT DETAILS

ARM	Details
PO.10	RS LCD PIN
P1.11	CE LCD PIN

REVIEW QUESTIONS:

1. Outline the operations involved when the key in a 4 x 4 keyboard matrix is being pressed.
2. List the registers used to store the keyboard, display modes and other operations programmed by CPU.
3. What is switch bouncing? How to prevent it using de-bounce circuit?
4. How to adjust the contrast of the LCD?
5. Which command of an LCD is used to shift the entire display to the right?

Result:The C-Language program for displaying the Key pressed in the Keyboard is displayed in the seven segment display and LCD module and the output was verified on the LCD on the desires line and column/address.

Ex. No. : 10

Date:

INTERFACING LED & PWM

Aim:

- a) To write a C program for Switch & LED to activate LED's
- b) To write a C program generate a PWM and to vary the duty cycle.

Apparatus & Software Required:

1. LPC2148 Development board.
2. Keil μ Vision5 software.
3. Flash Magic.
4. USB cable.
5. CRO.

Theory:

The PWM is based on the standard timer block and inherits all of its features, although only the PWM function is pinned out on the LPC2148. The timer is designed to count cycles of the peripheral clock (PCLK) and optionally generate interrupts or perform other actions when specified timer values occur, based on seven match registers. The PWM function is also based on match register events.

Procedure:

1. Follow the steps to create a New project
2. Type the below code and save it with the name (anyname.c)
3. Follow the steps to create a New Project to compile and build the program
4. Follow the procedures in to download your Hex code to processor using Flash Magic Software.

```

/*****
*****/

SWITCH AND LED PROGRAM
/*****
*****/
/* Description: This program gets DIP switch inputs and switches ON corresponding
LED
*/
/*          P1.16 to P1.31 are output switch          */

*****/

#include<LPC214x.H>

int main()
{

```

```

IO1DIR = 0xFFFF0000;           // P1.16 TO P1.31 OUTPUT PIN
while(1)
{
    IOCLR1 = 0xFFFF0000; // output pin cleared for enable the led
}
}

```

SWITCH AND LED PORT DETAILS

ARM	DETAILS
P1.16	S&L ENABLE PIN
P1.17	S&L ENABLE PIN
P1.18	S&L ENABLE PIN
P1.19	S&L ENABLE PIN
P1.20	S&L ENABLE PIN
P1.21	S&L ENABLE PIN
P1.22	S&L ENABLE PIN
P1.23	S&L ENABLE PIN
P1.24	S&L ENABLE PIN
P1.25	S&L ENABLE PIN
P1.26	S&L ENABLE PIN
P1.27	S&L ENABLE PIN
P1.28	S&L ENABLE PIN
P1.29	S&L ENABLE PIN
P1.30	S&L ENABLE PIN
P1.31	S&L ENABLE PIN

```

/*****
                *****/
                PWM.C
/*****
                *****/

/* Place lcd.c file into following directories C:\Keil\ARM\INC\Philips.*
/* This program is used to Generate the PWM, Frequency and Duty cycle can be
changed*/
*****

*****/

#include<LPC214x.H>

int main(void
{

PINSEL1 |= 0x00000400; //Enable pin0.7 as PWM2
PWMPR = 0x00000100; //Load prescaler (to vary the frequency can
modify here)

PWMPCR = 0x00002000; //PWM channel single edge control,
output enabled
PWMMCR = 0x00000003; //On match with timer reset the
counter

```

```

/* PWMR0 AND PWMR5 Both Value can change the duty cycle ex : PWMR0 = 10
AND PWMR5 = 2*/
PWMMR0 = 0x00000010; //set cycle rate to sixteen ticks
PWMMR5 = 0x00000008; //set rising edge of PWM2 to 2 ticks

PWMLER = 0x00000021; //enable shadow latch for
match 0 - 2
PWMTCR = 0x00000002; //Reset counter and
prescaler
PWMTCR = 0x00000009; //enable counter and PWM, release
counter from reset
while(1) // main loop
{
}
}

```

PWM PROGRAM PORT DETAIL

ARM	DETAILS
P0.7	PWM2

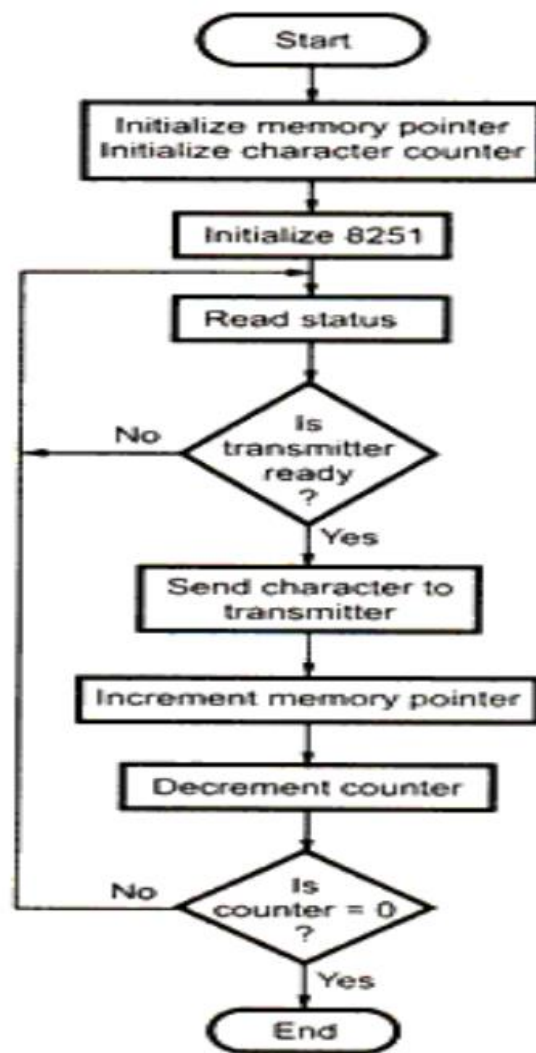
REVIEW QUESTIONS:

1. How do the variations in an average value get affected by PWM period?
2. Name the common formats available for LED display
3. Why are the pulse width modulated outputs required in most of the applications?
4. How do you determine the duty cycle of the waveform ?
5. What is the function of GPIO?

Result:

- a. The C code is generated for Switch & LED and output is verified in LED's by Switches.
- b. The C code is generated for PWM and to vary the duty cycle and verified in CRO output.

Flowchart




```

void SendI2CAddress(unsigned char Addr_S);
void WriteI2C(unsigned char
Data);
void StopI2C(void);
void StartI2C(void);

#define STA 0x20
#define SIC 0x08
#define SI 0x08
#define STO 0x10
#define STAC 0x20
#define AA 0x04

void InitI2C(void)
{
    I2C0CONCLR = 0xFF;
    PINSEL0 |= 0x50;    // Set pinouts as scl and sda
    I2C0SCLL = 19;     //speed at 100Khz for a VPB Clock Divider= 4 at 12 MHz
    I2C0SCLH = 19;
    I2C0CONSET = 0x40; //Active Master Mode on I2C bus
}

void SendI2CAddress(unsigned char Addr_S)
{
    while(I2C0STAT!=0x08);    // Wait for start to be
    completed
    I2C0DAT = Addr_S;        // Charge slave
    Address
    I2C0CONCLR = SIC | STAC; // Clear i2c interrupt bit to send
    the data
    while(!( I2C0CONSET & SI)) ; // wait till status available
}

unsigned char ReadI2C(void)
{
    unsigned char r;

    I2C0CONCLR = SIC;
    I2C0CONSET = 0x04;    // clear SIC;
    while(!(I2C0CONSET & 0x8)); // wait till status
    available r=I2C0STAT;
    wait();                // check for error
    if (r == 0x50){        // look for "Data byte has been
    received; ACK has been returned"
        lcdcmd(0x01);
        printstr("Read Sucess",0,0);
    }

    return I2C0DAT;
}

void WriteI2C(unsigned char Data)
{

```

```

unsigned char r;
I2C0DAT = Data;           // Charge Data
I2C0CONCLR = 0x8;        // SIC; Clear i2c interrupt bit to send
the data while(!(I2C0CONSET & 0x8)); // wait till status available
r=I2C0STAT;
if (r == 0x28)
    {
        // look for "Data byte in S1DAT has been
transmitted; ACK has been received"
        lcdcmd(0x01);
        printstr("Write Sucess",0,0);
    }
}

void StopI2C(void)
{
    I2C0CONCLR = SIC;
    I2C0CONSET = STO;
    while((I2C0CONSET&STO)); // wait for Stopped bus I2C
}

void StartI2C(void)
{
    I2C0CONCLR = 0xFF; // clear I2C - included if User
forgot to "StopI2C()"
    I2C0CONSET = 0x40; // else this function would hang.
// Active Master Mode on I2C bus
    I2C0CONSET = 0x00000020; // Start condition
}

int main()
{
    unsigned char r;

    wait();
    wait();
    wait();
    wait();
    lcdinit();
    clrscr();
    printstr("SM MICRRO SYSTEM",0,0);
    printstr("    ARM DEV KIT    ",0,1);
    InitI2C();
    StartI2C();
    SendI2CAddress(0xa0); // EEPROM device address
    WriteI2C(0); // Set the control port value
    WriteI2C('B');
    StopI2C();
    wait();
    wait();
    StartI2C();
    SendI2CAddress(0xa0); // EEPROM device address
    WriteI2C(0); // Set the control port value
    StopI2C();
    StartI2C();
}

```

```

SendI2CAddress(0xa1); // Start the read
r=ReadI2C(); // read the result StopI2C();
gotoxy(0,1);
split_numbers(r);
lcddat(0x30+hundreds);
lcddat(0x30+tens);
lcddat(0x30+ones);
while(1);

}

```

```

/*****

```

LCD.C

```

*****/

```

```

#define RS      0x00000400 /* P0.10 */
#define CE      0x00001800 /* P1.11 */
Void clrscr(char ch);
void lcdinit(void);
void lcdcmd(char); void lcddat(char);
void gotoxy(char,char); //x,y ; x-char position(0 - 16) y-line number 0
or 1
void printstr(char *,char,char); //string,column(x),line(y)
void wait (void);
void split_numbers(unsigned int number);
#define SET 1
#define OFF 0
unsigned int thousands,hundreds,tens,ones;
void wait (void) /* wait function */
int d;
for (d = 0; d < 100000; d++); /* only to delay for LED flashes */
}

void lcdinit()
{
IODIR0 = 0xFFFFFFF;
IOCLR0 = 0X00000FFF;
lcdcmd(0x28);
lcdcmd(0x28);
lcdcmd(0x0c);
lcdcmd(0x06);
lcdcmd(0x01);
lcdcmd(0x0f);
wait();//(1600);
}

void gotoxy(char x, char y)
{
if(y == 0)
lcdcmd(0x80+x);
else
lcdcmd(0xc0+x);
}

```



```

}
void printstr(char *str, char x, char y)
{
    char i; gotoxy(x,y); wait();//(500);
    for(i=0;str[i]!='\0';i++) lcddat(str[i]);

    }
void lcdcmd(char cmd)
{
    unsigned char LCDDAT;
    LCDDAT = (cmd & 0xf0); //higher nibble IOSET0 = LCDDAT;
    IOCLR0 = RS; IOSET0 = CE;
    wait();//(100); //enable lcd
    IOCLR0 = CE;
    IOCLR0 = 0X00000FFF;
    LCDDAT = ((cmd<<0x04) & 0xf0); //lower nibble IOSET0 = LCDDAT;
    IOCLR0 = RS; IOSET0 = CE;
    wait();//(100); //enable lcd
    IOCLR0 = CE;
    IOCLR0 = 0X00000FFF;

}

void lcddat(char cmd)
{
    unsigned char LCDDAT;
    LCDDAT = (cmd & 0xf0); //higher nibble IOSET0 = LCDDAT;
    IOSET0 = RS; IOSET0 = CE;
    wait();//(100); //enable lcd
    IOCLR0 = CE;
    IOCLR0 = 0X00000FFF;
    LCDDAT = ((cmd<<0x04) & 0xf0); //lower nibble IOSET0 = LCDDAT;
    IOSET0 = RS; IOSET0 = CE;
    wait();//(100); //enable lcd
    IOCLR0 = CE;
    IOCLR0 = 0X00000FFF;
}
void clrscr(char ch)
{
    if(ch==0)
    {
        printstr(" ",0,0);
        gotoxy(0,0);
    }
    else if(ch == 1)
    {
        printstr(" ",0,1);
        gotoxy(0,1);
    }
    else
    {
        lcdcmd(0
        x01);
    }
}

```

```

    //
    delay(100);
  }
}

void split_numbers(unsigned int number)
{
  thousands =
  (number
  /1000);
  number %=
  1000;
  hundreds =
  (number /
  100);
  number %=
  100;
  tens =
  (numbe
  r / 10);
  number
  %= 10;
  ones = number;
}

```

EPROM (I2C) PROGRAM PORT SETAILS

ARM	DETAILS
PO.10	RS LCD PIN
P1.11	CE LCD PIN
P0.11	SCL
P0.14	SDA

INTERRUPT BUZZER PROGRAM

```

/*****
***/
Ext Driver.C
/*****/

#include
de
<LP
C21
4x.h
>
void
init_
VIC(
void)

```

```

{
    /*
    initialize
    VIC*/
    VICIntEnClr = 0xffffffff; VICVectAddr    = 0;
    VICIntSelect = 0;
}

void ExtInt_ISR(void)_irq
{
    //EXTINT = (1<<2); /* clear EINT2 flag by writing HIGH to coresponding bit*/
    //IOCLR0 = 0x40000000; /* Trigger the relay*/
    IOCLR1 = 0x400f0000; /* P1.18 Trigger the relay*/
    //IOPIN
    1          =
    0x00000000;
    EXTINT     =
    (1<<2);
    VICVectAddr = 0; /* Acknowledge Interrupt */
}
void init_Interrupt(void)
{
    PINSEL0 = 0x80000000; // select P0.15 for
    EINT2
    VICIntEnable = (1 << 16); // External
    interrupt 2 (EINT2)
    VICVectCntl0 = (1<<5)|(16); // set the VIC control reg
    for EINT2 VICVectAddr0 = (unsigned long)ExtInt_ISR;
    EXTMODE &= ~(1<<2); // set VIC for egdse sensitive for EINT2
// EXTPOLAR = ~(1<<2); // set VIC for falling edge sensitive for EINT2
}
void init_ports(void)
{
    IODI
    R0 =
    0x400
    00000
    ;
    IODI
    R1 =
    0x400
    f0000
    ;
    IOPI
    N1 =
    0xff0
    10000
    ;
    IOSE
    T0 =
    0x400
    00000
    ;
    IOSET1 = 0x400f0000;
}

```

```

    }
    /*void wait_for_turnoffRelay(void)
    {
        int val;
        val = IOPIN1;           // read the ports for key board
        input while((~(val>>20)) != 0); // wait until 1st key in the matrix
        keyboard
    is pressed
        IOCLR0 = 0x00010000;    // switch off the relay
    }*/

/*****
*****/

```

XINTR_RELAY.C

```

*****/

#include <LPC214x.h>
#include "ext.h"
int main()
{
    init_VIC();
    init_Interrupt();
    init_ports();
    while(1)
    {
        //wait_for_turnoffRelay();
    }
}

```

INTERRUPT BUZZER PROGRAM

ARM	DETAILS
P1.18	TRIGGER THE RELAY
P0.15	EINT2

Review Questions

1. What will be the initial values in all the cells of an EPROM?
2. What are the contents of the IE register, when the interrupt of the memory location 0x00 is caused?
3. Why normally LJMP instructions are the topmost lines of the ISR?
4. Enumerate the features of nested interrupt.
5. Illustrate the Master Slave mode.

Result:

The C-Language program to write and read a data in EEPROM and also to analyze its performance with the interrupt is developed and is verified.

Ex. No. 11

Date:

ANALYZE SERIAL INTERFACE AND PARALLEL INTERFACE

Aim:

To write an ALP to demonstrate

- (a) Serial Interface - transmit a data 41H serially by interfacing 8086 with 8251
- (b) Parallel Interface

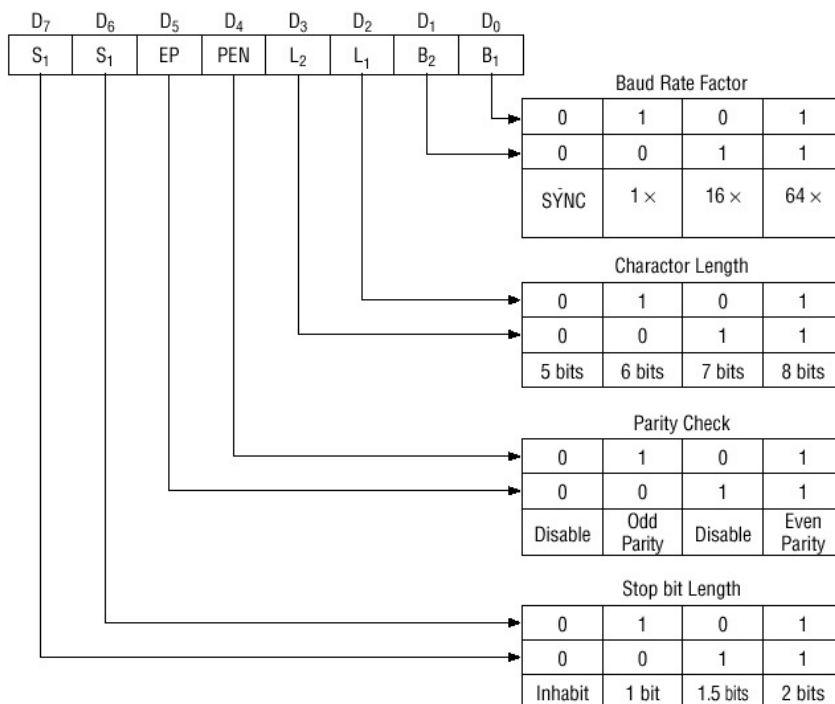
SERIAL INTERFACE

Description:

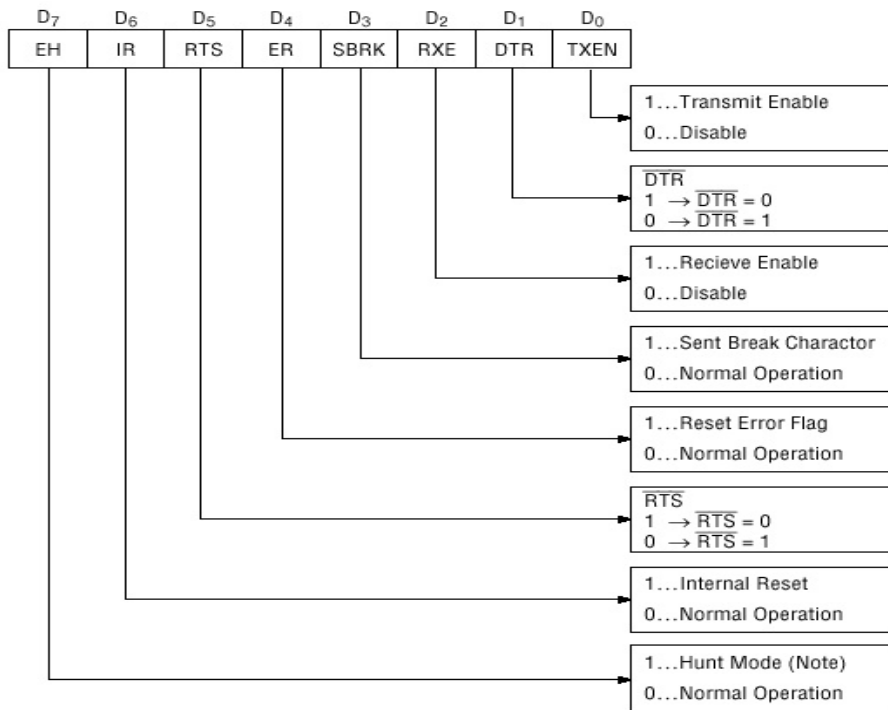
The 8253 and 8251 should be initialized before transmitting the character. The Program first initialize 8253 to give an output clock frequency of 150 KHz at channel 0 which will give a 9600 baud rate of 8251. The 8251 mode instruction (refer mode instruction format) is initialized with the following specifications: 8bit data, No parity, Baud rate factor (16x), 1 stop bit. Thus the mode command word is 4E for the above said specifications. The 8251 command instruction (refer command instruction format) is initialized with 37H which enables the transmit enable and receive enable bits, force DTR output to zero, resets the error flags, and forces RTS output to zero.

Algorithm:

1. Start the program.
2. Set the origin as 1100H.
3. Initialize the 8253 Timer in Mode 3
4. Initialize the 8251
5. Transmit the data at transmitter end
6. Reset the system
7. At the receiver end receive the data and reset the system
8. Stop the program.



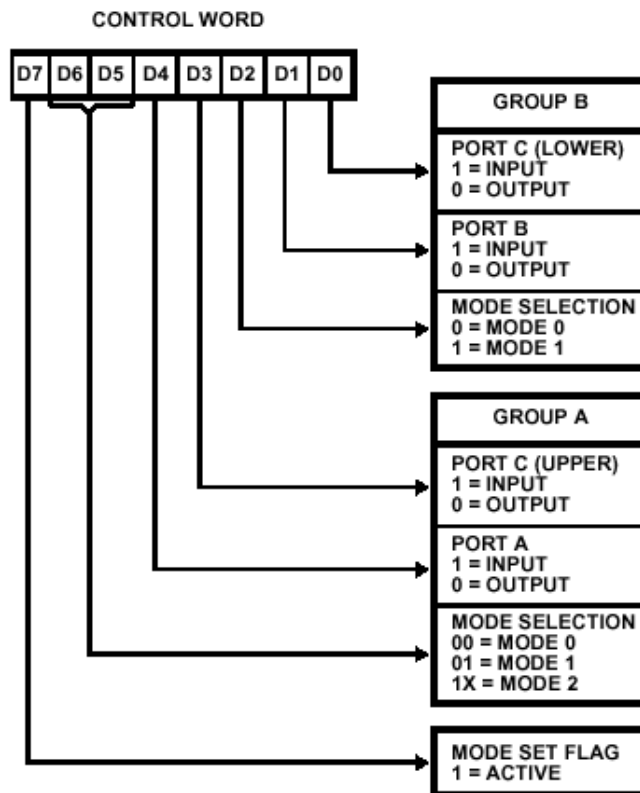
Bit Configuration of Mode Instruction (Asynchronous)



Note: Search mode for synchronous characters in synchronous mode.

Bit Configuration of Command

CONTROL WORD FORMAT OF 8255



PROGRAM:

Label	Program	Comments
	ORG 1000H	Set starting address as 1000H.
	MOV AL, 36	Mode set for 8253 – Channel 0 in Mode 3
	OUT CE, AL	
	MOV AL, 10	
	OUT C8, AL	
	MOV AL, 00	
	OUT C8, AL	
	MOV AL, 4E	Mode instruction for 8251
	OUT C2, AL	
	MOV AL, 37	Command Instruction for 8251
	OUT C2, AL	
	MOV AL, 41	
	OUT C0, AL	Sent the data 41
	INT 2	Reset
	ORG 1200H	
	IN AL,C0	Receive the data 41
	MOV BX,1250	
	MOV [BX],AL	Store the data at 1250H
	INT 2	Reset

Observation:

Output:
1250:

REVIEW QUESTIONS:

1. Expand USART?
2. Where do we prefer the serial communication?
3. What is the function of instruction pointer (IP) register?
4. What is the difference between IN and OUT instructions?
5. What is MODEM?

PARALLEL INTERFACE**Description:**

Initialize the Port A as Input port and Port B as Output port in Mode – 0. The input port reads the data set by the SPDT switches and the output port outputs the same data to port B to glow LEDs accordingly.

Algorithm:

1. Start the program.
2. Set the origin as 1100H.
3. Initialize the port A as input port
4. Initialize the port B as output port
5. Configure 8255 in mode 0
6. Read the input port
7. Write the read data to the output port
8. Stop the program.

Parallel Interface Program

Label	Program	Comments
	ORG 1100H	Set starting address as 1100H.
	MOV AL,90	Initialize 8255 in mode 0 with port A as input port and port B as output port.
	OUT C6,AL	
	IN AL,C0	Read the data from SPDT switch
	OUT C2,AL	Write the data to LEDs
	HLT	

Example:**Input:**

SPDT switch position: 10110011

Output:

LED status: 10110011

Manual Calculation:**REVIEW QUESTIONS:**

1. What is the difference between near and far procedure?
2. What is difference between shifts and rotate instructions?
3. Which are strings related instructions?
4. Which are addressing modes and their examples in 8086?
5. Discuss the use of following instructions:
 - a. SCASB
 - b. LAHF
 - c. ROL
 - d. SHR
 - e. IDIV

Result:

Thus the programs for serial and parallel interface are executed successfully.

Ex. No. 12

Date:

A/D AND D/A INTERFACE

Aim:

To write an assembly language program to demonstrate

(a) Analog to Digital Conversion

(b) Digital to Analog Conversion

ANALOG TO DIGITAL CONVERSION

Features of ADC 0809

ADC 0809 is a monolithic CMOS device, with an 8-bit analog to digital converter, 8 channel multiplexer and microprocessor compatible control logic

1. 8 bit resolution
2. 100 μ s Conversion time
3. 8 channel multiplexer with latched control logic
4. No need for external zero or full scale adjustments
5. Low power consumption time
6. Latched tristate output

The device contains an 8 channel single ended analog signal multiplexer. A particular input channel. A particular input channel is selected by using the address decoding. Table shows the input states for the address lines to select any channel. The address is latched into the decoder of the chip on low to high transition of the address latch enable. The A/D converter's successive approximation register reset on the positive edge of the start of the conversion pulse. The conversion is begun on the falling edge of the SOC pulse. End of conversion will go low between 0 and 8 clock pulses after the rising edge of start of conversion

SELECTED ANALOG CHANNEL	ADDRESS LINE		
	ADD C	ADD B	ADD A
IN0	0	0	0
IN1	0	0	1
IN2	0	1	0
IN3	0	1	1
IN4	1	0	0
IN5	1	0	1
IN6	1	1	0
IN7	1	1	1

Algorithm

1. Select Channel '0' and apply analog voltage
2. Send Start of conversion
3. Check End of conversion
4. Get digital data for corresponding analog voltage and display at stored location.

The buffer 74LS244 which transfers the converted data outputs to data bus is selected when

A7	A6	A5	A4	A3	A2	A1	A0	
1	1	0	0	0	X	X	X	=C0H

The I/O address for the latch 74LS 714 which latches the data bus to ADD A, ADD B and ADDC and ALE 1 and ALE 2 is

A7	A6	A5	A4	A3	A2	A1	A0	
1	1	0	0	1	X	X	X	=C8H

The flip flop 74LS74 which transfers the D0 line status to the start of conversion pin of ADC0809 is selected when

A7	A6	A5	A4	A3	A2	A1	A0	
1	1	0	1	0	X	X	X	=D0H

The EOC output of ADC 1 and ADC 2 is transferred to D0 line by means of two tristate buffers.

The EOC 1 is selected when

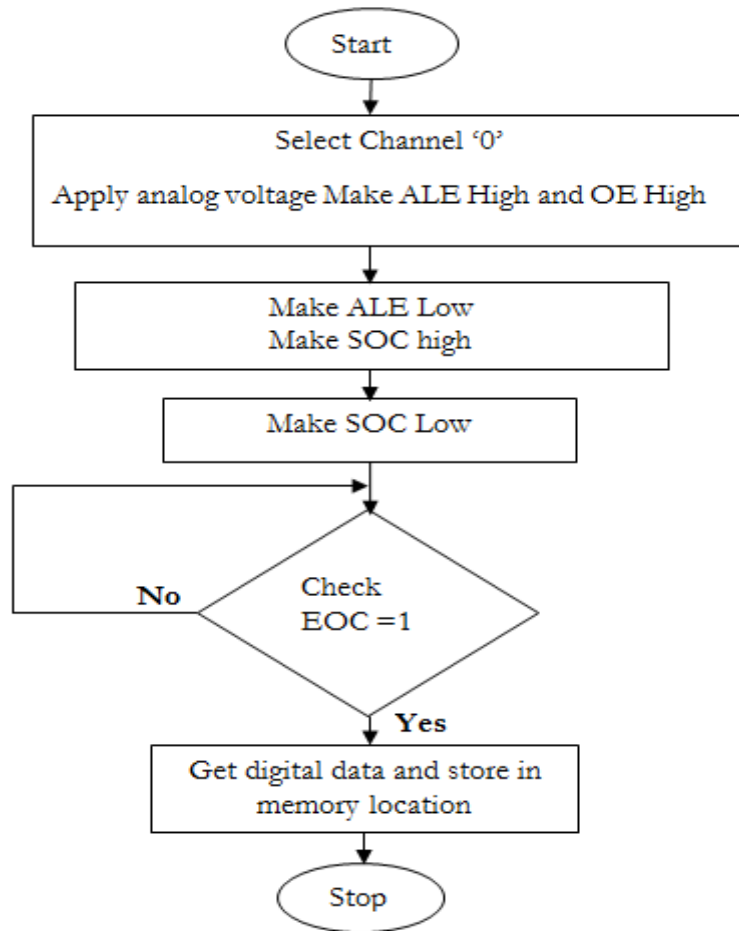
A7	A6	A5	A4	A3	A2	A1	A0	=D8H
1	1	0	1	1	X	X	X	

The EOC 2 is selected when

A7	A6	A5	A4	A3	A2	A1	A0	=E0H
1	1	1	0	0	X	X	X	

SL. NO	CHANNEL NUMBER	EOC ADDRESS	CHNO. ALE LOW OE HIGH	CHNO. ALE HIGH OE LOW	CHNO. ALE LOW OE HIGH
1	CH0	D8	10	18	10
2	CH1	D8	11	19	11
3	CH2	D8	12	1A	12
4	CH3	D8	13	1B	13
5	CH4	D8	14	1C	14
6	CH5	D8	15	1D	15
7	CH6	D8	16	1E	16
8	CH7	D8	17	1F	17

FLOWCHART



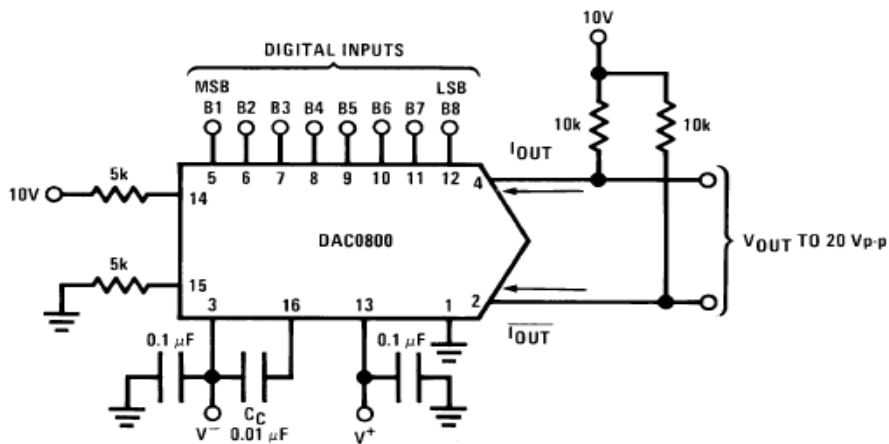
PROGRAM

Label	Program	Comments
	ORG 4100H	Set starting address as 4100H.
	MOV AL, 10H	Selection Channel '0'
	OUT 0C8H, AL	
	MOV AL, 18H	Make ALE1 and OE1 high
	OUT 0C8H, AL	
	MOV AL, 01H	Make SOC High
	OUT 0DOH, AL	
	MOV AL, 00H	
	MOV AL, 00H	
	MOV AL, 00H	
	MOV AL, 00H	Make SOC low
	OUT 0DOH, AL	
LOOP	IN AL, 0D8H	Check EOC
	AND AL, 01H	
	CMP AL, 01H	

	JNZ LOOP IN AL, 0C0 MOV BX, 1200H MOV [BX], AL HLT	Output Digital Data
--	--	---------------------

Observation:

DAC 0800



REVIEW QUESTIONS:

1. Which is by default pointer for CS/ES?
2. What is the difference between instructions RET & IRET?
3. What are the functions performed by 8279?
4. What is PPI?
5. Give the control word format for I/O mode of 8255?

INTERFACING DAC WITH 8086

THEORY:

DAC 0800 is an 8 – bit DAC and the output voltage variation is between – 5V and + 5V. The output voltage varies in steps of $10/256 = 0.04$ (appx.). The digital data input and the corresponding output voltages are presented in the Table1.

Input Data in HEX	Output Voltage
00	- 5.00
01	- 4.96
02	- 4.92
...	...
7F	0.00
...	...

FD	4.92
FE	4.96
FF	5.00

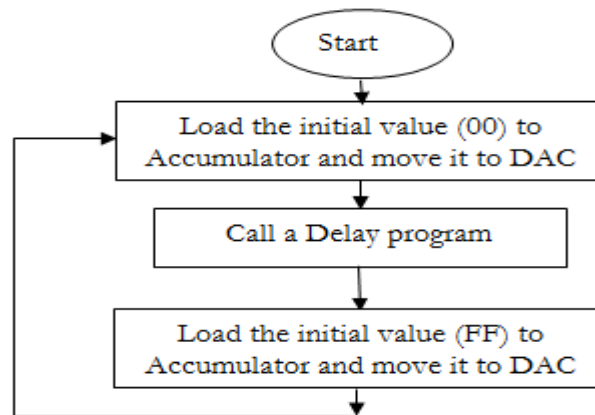
Referring to Table1, with 00 H as input to DAC, the analog output is – 5V. Similarly, with FF H as input, the output is +5V. Outputting digital data 00 and FF at regular intervals, to DAC, results in different wave forms namely square, triangular, etc.. The port address of DAC is 08 H

ALGORITHM:

(a) Square Wave Generation

1. Load the initial value (00) to Accumulator and move it to DAC
2. Call the delay program
3. Load the final value(FF) to accumulator and move it to DAC
4. Call the delay program.
5. Repeat Steps 2 to 5

FLOWCHART



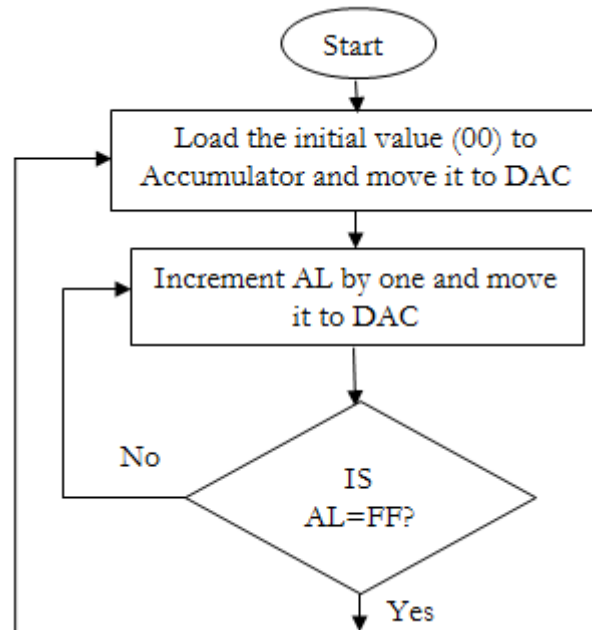
PROGRAM

Label	Program	Comments
START:	<pre> ORG 4100H MOV AL, 00H OUT 0C0H,AL CALL DELAY MOV AL, 0FFH OUT 0C0H,AL CALL DELAY JMP START </pre>	Set starting address as 4100H.
DELAY: L1:	<pre> MOV CX, 05FFH LOOP L1 RET </pre>	

(b) Saw tooth Wave Generation

1. Load the initial value (00) to Accumulator
2. Move the accumulator content to DAC
3. Increment the accumulator content by 1.
4. Repeat Steps 3 and 4.

FLOWCHART



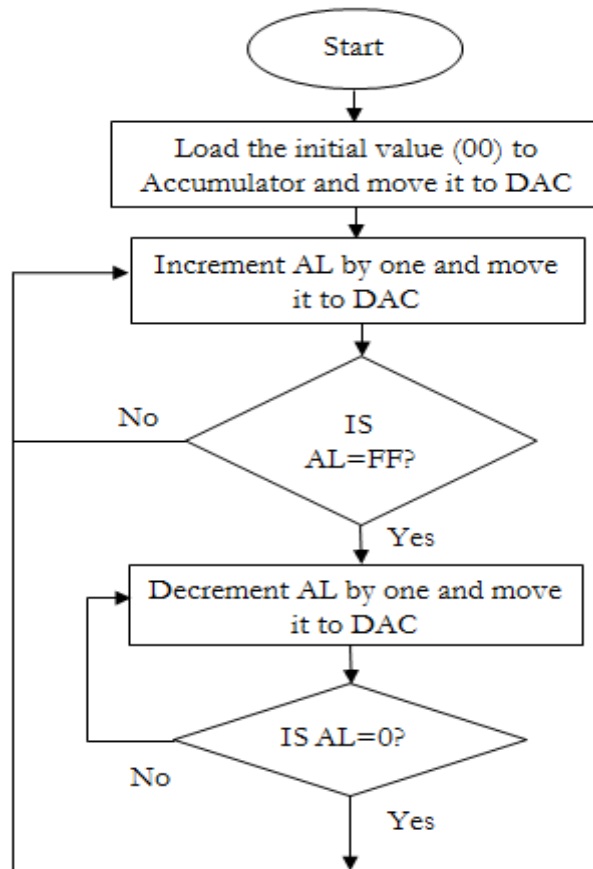
PROGRAM

Label	Program	Comments
START L1	ORG 4100H MOV AL, 00H OUT 0C0H, AL INC AL JNZ L1 JMP START	Set starting address as 4100H.

(c) Triangular Wave Generation

1. Load the initial value (00) to Accumulator
2. Move the accumulator content to DAC
3. Increment the accumulator content by 1.
4. If accumulator content is zero proceed to next step. Else go to step 3.
5. Load value (FF) to Accumulator
6. Move the accumulator content to DAC
7. Decrement the accumulator content by 1.
8. If accumulator content is zero go to step2. Else go to step 7.

FLOWCHART



PROGRAM

Label	Program	Comments
	ORG 4100H	Set starting address as 4100H.
START:	MOV BL, 00H	
L1:	MOV AL, BL	
	OUT 0C0H,AL	
	INC BL	
	JNZ L1	
	MOV BL, 0FFH	
L2:	MOV AL, BL	
	OUT 0C0H,AL	
	DEC BL	
	JNZ L2	
	JMP START	

Example:

Waveform	Amplitude	Time Period(ms)
Square	2	56
Sawtooth	2	3
Triangular	2	2.4

Observation:

Waveform	Amplitude	Time Period(ms)

Square		
Sawtooth		
Triangular		

REVIEW QUESTIONS:

1. Whether 8086 is compatible with Pentium processor?
2. Write an ALP program for multiplication of given number in location mode a) 0060, b) 0002
3. List the operating modes of 8253 timer.
4. What is the use of USART?
5. Compare the serial and parallel communications.

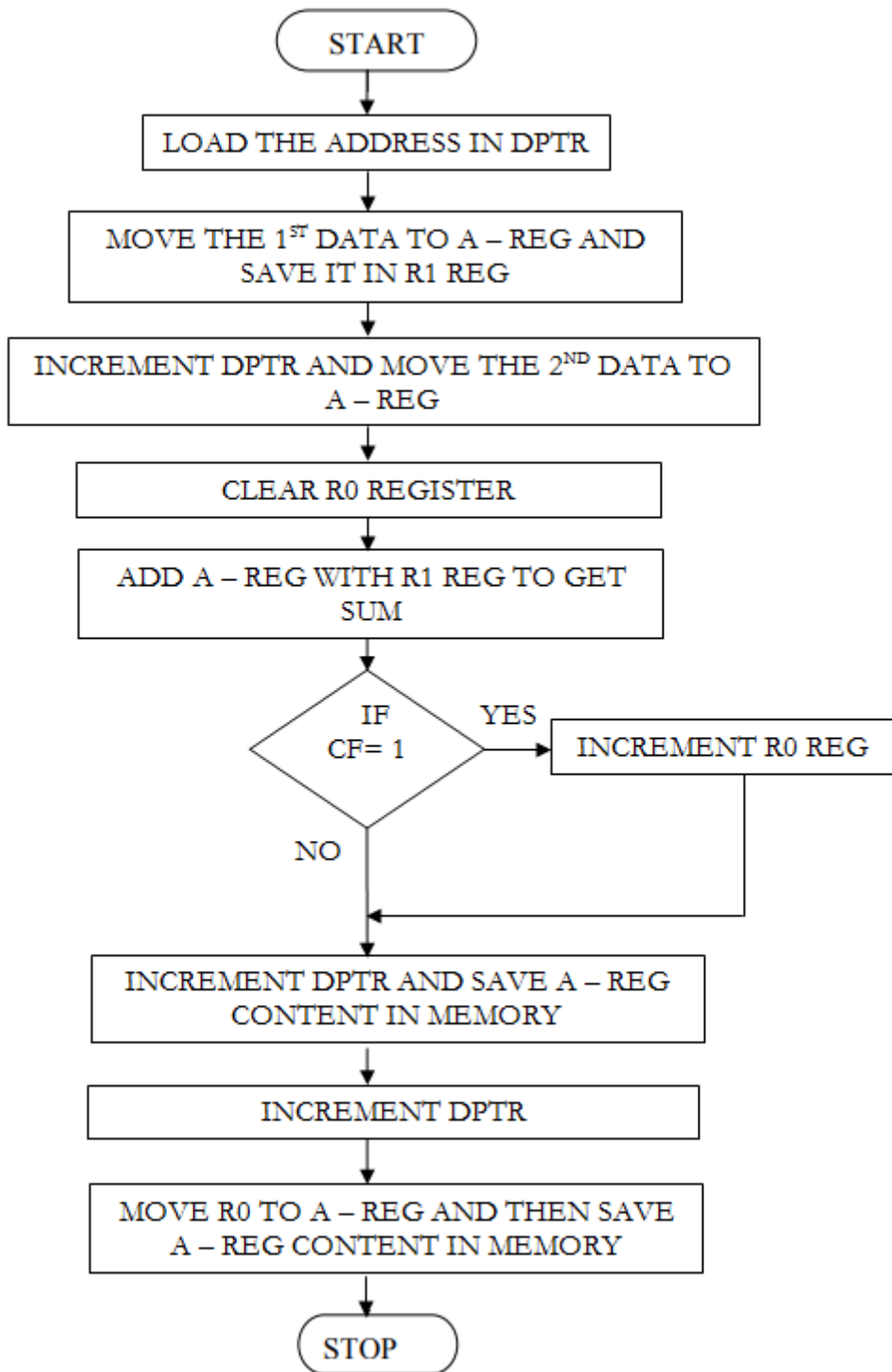
RESULT

Thus the program to demonstrate the ADC and DAC were executed.

CYCLE III

8051 Programs

Flow Chart



Ex. No. 13**Date:****BASIC ARITHMETIC AND LOGIC OPERATIONS****Objective:**

To write an ALP to perform the following operations using 8051 instruction set

- (a) Addition
- (b) Subtraction
- (c) Multiplication
- (d) Division
- (e) Logical operation

ADDITION OF TWO 8 BIT NUMBERS**Description:**

In order to perform addition in 8051, one of the data should be in accumulator and another data can be in any SFR/internal RAM or can be an immediate data. After addition the sum is stored in accumulator. The sum of two 8 – bit data can be either 8 bits (sum only) or 9 bits (sum and carry). The accumulator can accommodate only the sum and if there is carry, the 8051 will indicate by setting carry flag. Hence one of the internal register/RAM locations can be used to account for carry.

Algorithm:

1. Set DPTR as pointer for data.
2. Move first data from external memory to accumulator and save it in R1 register.
3. Increment DPTR.
4. Move second data from external memory to accumulator
5. Clear R0 register to account for carry.
6. Add the content of R1 register to accumulator.
7. Check for carry. If carry is not set go to step 8. Otherwise go to next step.
8. Increment R0 register.
9. Increment DPTR and save the sum in external memory.
10. Increment DPTR, move carry to accumulator and save it in external memory.
11. Stop

PROGRAM:

Label	Program	Comments
	MOV DPTR,#4500	Load address of 1 st data in DPTR
	MOVX A,@DPTR	Move the 1 st data to A
	MOV R1,A	Save the first data in R1
	INC DPTR	Increment DPTR to point 2 nd data
	MOVX A,@DPTR	Load the 2 nd data in A
	MOV R0,#00	Clear R0 for the account of carry
	ADD A,R1	Get the sum in A reg
	JNC AHEAD	Check carry flag
	INC R0	If carry is set increment R0
AHEAD:	INC DPTR	Increment DPTR
	MOVX @DPTR,A	Save the sum in external memory
	INC DPTR	Increment DPTR

HERE:	MOV A,R0 MOVX @DPTR,A SJMP HERE	Move carry to A reg Save the carry in external memory Remain idle in infinite loop
-------	---------------------------------------	--

Example:

Manual Calculation:

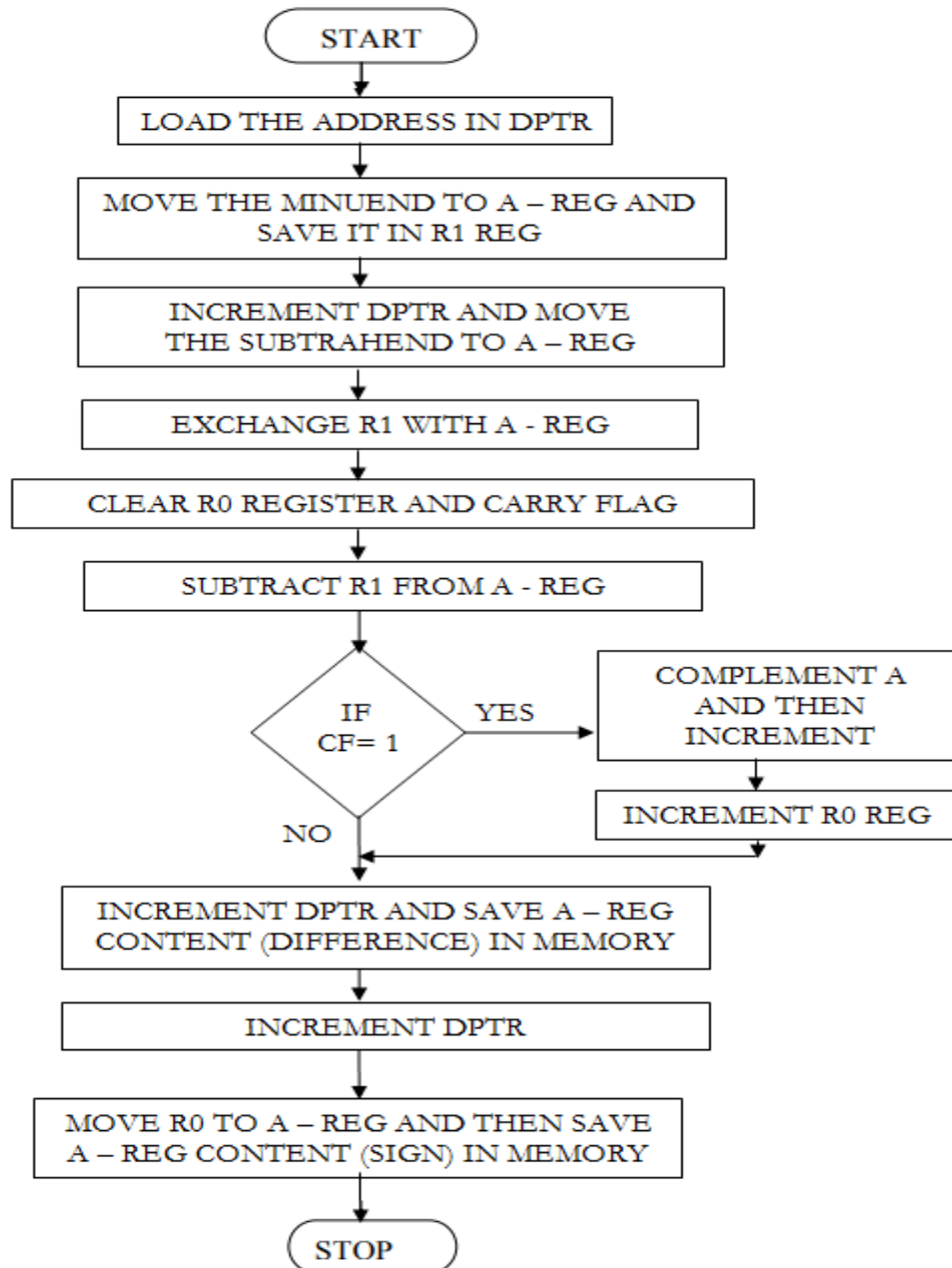
Input:

4500: 05 [Addend]
 4501: 06 [Augend]

Output:

4502: 0B [Sum]
 4503: 00 [Carry]

Flow Chart



SUBTRACTION OF TWO 8 BIT NUMBERS

Description:

In order to perform subtraction in 8051, one of the data should be in accumulator and another data can be in any SFR/internal RAM or can be an immediate data. After subtraction the result is stored in accumulator. The 8051 perform 2's complement subtraction and then complement the carry. Therefore if the result is negative carry flag is set and the accumulator will have 2's complement of the result. In order to get the magnitude of the result again take 2's complement of the result. One of the register is used to account for the sign of the result. The 8051 will consider previous carry while performing subtraction and so the carry should be cleared before performing subtraction.

Algorithm:

1. Set DPTR as pointer for data.

2. Move the minuend from external memory to accumulator and save it in R1 register.
3. Increment DPTR.
4. Move subtrahend from external memory to accumulator
5. Exchange the contents of R1 and A such that minuend is in A and subtrahend is in R1
6. Clear R0 register to account for sign.
7. Clear carry flag.
8. Subtract the content of R1 register from accumulator.
9. Check for carry. If carry is not set go to step 12. Otherwise go to next step.
10. Complement the content of A – reg and increment by 1 to get 2's complement of result in A – reg
11. Increment R0 register.
12. Increment DPTR and save the result in external memory.
13. Increment DPTR, move R0 (sign bit) to accumulator and then save it in external memory.
14. Stop

PROGRAM:

Label	Program	Comments
	MOV DPTR,#4500	Load address of minuend in DPTR
	MOVX A,@DPTR	Move the minuend to A
	MOV R1,A	Save the minuend in R1
	INC DPTR	Increment DPTR to point subtrahend
	MOVX A,@DPTR	Load the subtrahend in A
	XCH A,R1	Get minuend in A and Subtrahend in R1
	MOV R0,#00	Clear R0 for the account of Sign
	CLR C	Clear carry
	SUBB A,R1	Subtract R1 from A
	JNC AHEAD	Check Carry flag. If carry is set then
	CPL A	Get 2's complement of result in A
	INC A	
	INC R0	Set R0 to indicate negative sign
AHEAD:	INC DPTR	Increment DPTR
	MOVX @DPTR,A	Save the result in external memory
	INC DPTR	Increment DPTR
	MOV A,R0	Move sign bit to A reg
	MOVX @DPTR,A	Save the sign in external memory
HERE:	SJMP HERE	Remain idle in infinite loop

Example:

Input:

4500: 0A

4501:05

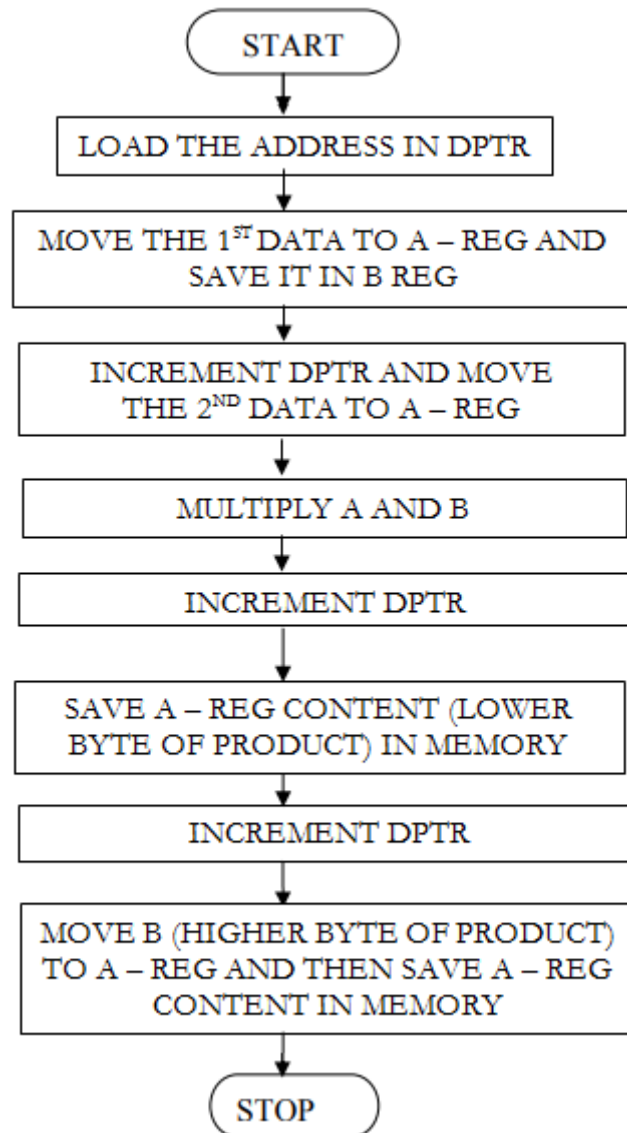
[Minuend]

[Subtrahend]

Manual Calculation:

Output:

4502:05 [Difference]
4503:00 [Sign Bit]

Flow Chart**MULTIPLICATION OF TWO 8 BIT NUMBERS****Objective:**

To write an ALP to multiply two numbers of 8-bit data using 8051 instruction set

Description:

In order to perform subtraction in 8051, the two 8 – bit data should be stored in A and B registers, then multiplication can be performed by using “MUL AB” instruction. After multiplication the 16 – bit product will be in A and B register such that lower byte in A and higher byte in B register.

Algorithm:

1. Load address of data in DPTR
2. Move the first data from external memory to A and save in B.
3. Increment DPTR and move second data from external memory to B.

4. Perform multiplication to get the product in A and B.
5. Increment DPTR and save A (lower byte of product) in memory
6. Increment DPTR , move B (lower byte of product) to A and save it in memory
7. Stop

PROGRAM:

Label	Program	Comments
	MOV DPTR,#4500	Load address of 1 st data in DPTR
	MOVX A,@DPTR	Move the 1 st data to A
	MOV B,A	Save the 1 st data in B
	INC DPTR	Increment DPTR to point 2 nd data
	MOVX A,@DPTR	Load the 2 nd data in A
	MUL AB	Get the product in A and B
	INC DPTR	Increment DPTR
	MOVX @DPTR,A	Save the lower byte of result in external memory
	INC DPTR	Increment DPTR
	MOV A,B	Move the higher byte of product to A reg
	MOVX @DPTR,A	Save it in external memory
HERE:	SJMP HERE	Remain idle in infinite loop

Example:

Input:

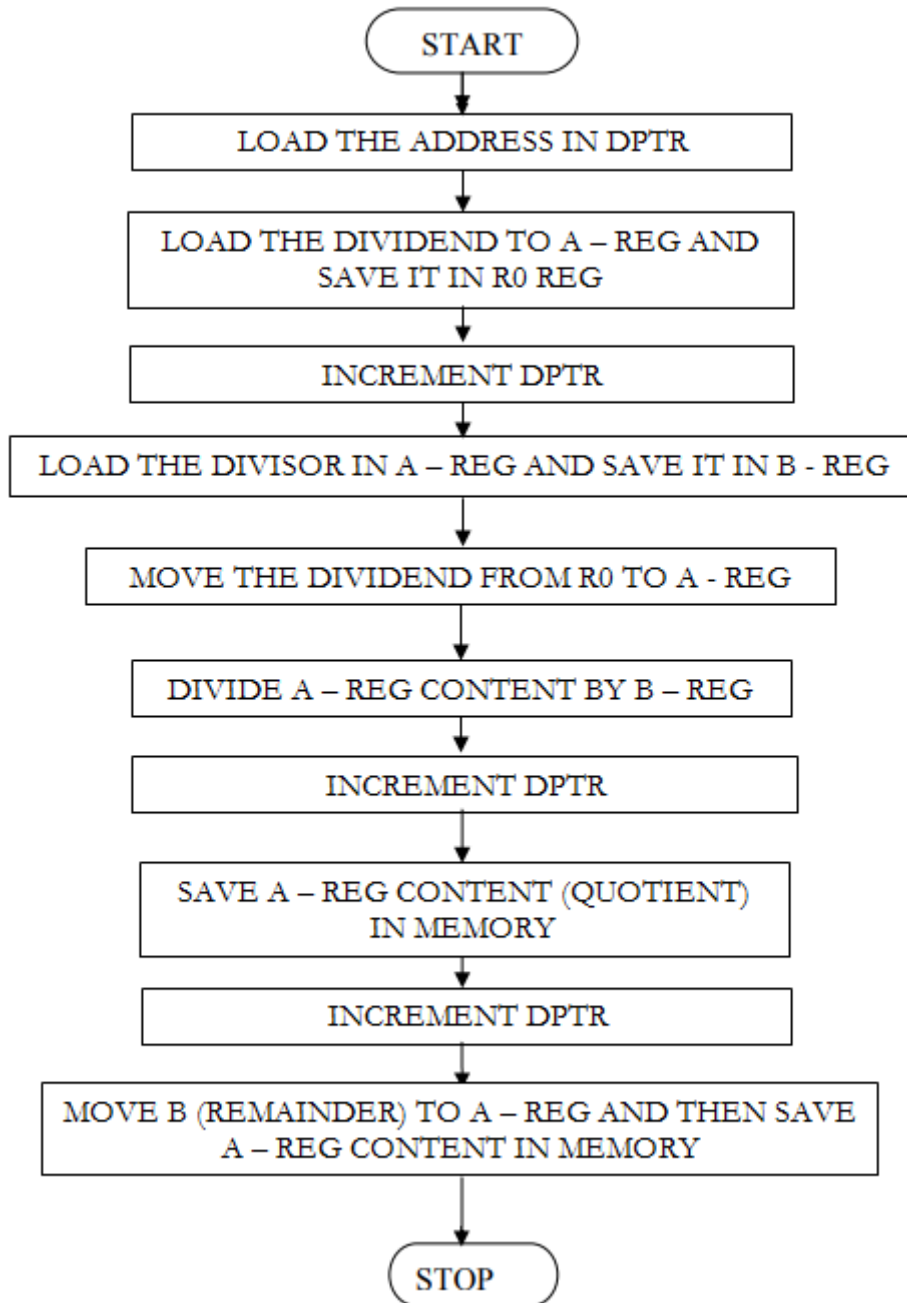
4500:02 [1st data]
 4501:03 [2nd data]

Output:

4502:06 [Lower byte of product]
 4503:00 [Higher byte of product]

Manual Calculation:

FLOWCHART



DIVISION OF TWO 8 BIT NUMBERS

Description:

In order to perform subtraction in 8051, the dividend should be stored in A – reg and divisor should be stored in B – reg. then the content of A can be divided by B using the instruction “DIV AB”. After division the quotient will be in A – reg and remainder will be in B – reg.

Algorithm:

1. Load address of data in DPTR
2. Move the dividend from external memory to A and save it in R0 register.
3. Increment DPTR and move the divisor from external memory to A and save it in B reg.
4. Move the dividend from R0 to A.
5. Perform division to get quotient in A and remainder in B.
6. Increment DPTR and save quotient (content of A - reg) in memory
7. Increment DPTR.

8. Move the remainder (Content of B – reg) to A and save in memory.
9. Stop

Label	Program	Comments
	MOV DPTR,#4500	Load address of dividend in DPTR
	MOVX A,@DPTR	Move the dividend to A
	MOV R0,A	Save the dividend in R0
	INC DPTR	Increment DPTR to point divisor
	MOVX A,@DPTR	Load the divisor in A
	MOV B,A	Move the divisor to B
	MOV A,R0	Move the dividend to A
	DIV AB	Divide the content of A by B
	INC DPTR	Increment DPTR
	MOVX @DPTR,A	Save the quotient in external memory
	INC DPTR	Increment DPTR
	MOV A,B	Move the remainder to A reg
HERE:	MOVX @DPTR,A	Save it in external memory
	SJMP HERE	Remain idle in infinite loop

Example:

Input:

4500: 04

[Dividend]

4501: 02

[Divisor]

Output:

4502: 02

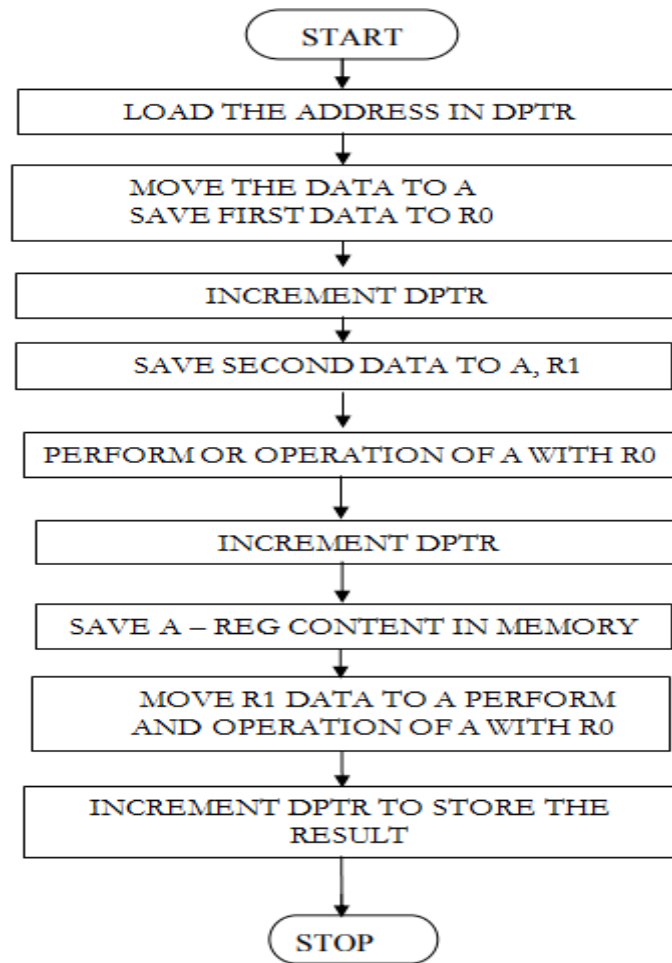
[Quotient]

4503: 00

[Remainder]

Manual Calculation:

FLOWCHART



LOGICAL OPERATIONS OF 8 BIT NUMBERS

Description:

The first value should be stored in R0 -reg, second value should be stored in R1 – reg, First move R1 value to A, perform OR operation with R0 reg and store the result. Second move R1 value to A performs AND operation with R0 reg stores the result.

Algorithm:

1. Load address of first data in DPTR
2. Move the data to A
3. Save first data to R0
4. Increment DPTR to Load address of second data in DPTR
5. Save second data to A, R1
6. Perform OR operation of A with R0
7. Increment DPTR to store the result
8. Move R1 data to A
9. Perform AND operation of A with R0
10. Increment DPTR to store the result

PROGRAM:

Label	Program	Comments
	MOV DPTR,#4500	Load address of first data in DPTR
	MOVX A,@DPTR	Move the data to A
	MOV R0, A	Save first data to R0
	INC DPTR	Increment DPTR to Load address of second data in DPTR
	MOVX A,@DPTR	
	MOV R1,A	Save second data to A, R1
	ORL A, R0	Perform OR operation
	INC DPTR	Increment DPTR to store the result
	MOVX @DPTR, A	
	MOV A, R1	
	ANL A, R0	Perform AND operation
	INC DPTR	Increment DPTR to store the result
	MOVX @DPTR, A	
HERE:	SJMP HERE	

Example:**Input**

4500 :00

4501:01

Output

4502 :01 (OR operation)

4503 :00 (AND operation)

Manual Calculation:**PROGRAM:**

Label	Program	Comments
	ORG 4100H	Set starting address as 4100H.
	MOV DPTR, #4500H	Initialise the dptr
	MOVX A,@DPTR	Get the data in A – reg
	MOV B,A	Copy it in B – reg
	MUL AB	Multiply A and B
	INC DPTR	Increment dptr
	MOVX @DPTR,A	Store the lower order in memory
	INC DPTR	Increment dptr
	MOV A,B	

HERE:	MOVX @DPTR,A SJMP HERE	Store the higher order in memory
-------	---------------------------	----------------------------------

Example:

Input:

4500:03

Output:

4501:09

4502:00

REVIEW QUESTIONS:

1. What is a microcontroller? How does it differ from a microprocessor?
2. What is the role of the program counter in 8051?
3. Write the significance of oscillators in a microcontroller.
4. What are the types of memory in 8051?
5. What is PSW?
6. Draw the format of TMOD register.

Result:

Thus the program for arithmetic and logic operation was written and executed.

Ex. No. 14

Date:

SQUARE, CUBE and 2'S COMPLIMENT OF A NUMBER

Objective:

To write 8051 ALP to determine the square, cube and 2's compliment of a number

SQUARE OF A NUMBER

Description:

The square of a number is determined by multiplying the value by itself. In this program the input is obtained in A – reg and then it is copied to B – reg. The values of A and B registers are multiplied and the result is stored in memory.

Algorithm:

1. Start the program.
2. Set the origin as 4500H.
3. Initialize DPTR
4. Get the value in A – reg and copy it in B – reg
5. Multiply the values of A – reg and B – reg
6. Store the result
7. Stop the program.

PROGRAM:

Label	Program	Comments
	ORG 4100H	Set starting address as 4100H.
	MOV DPTR,#4500H	Initialise the dptr
	MOVX A,@DPTR	Get the data in A – reg
	MOV R0,A	Copy it in r0 – reg
	MOV B,A	Copy it in B – reg
	MUL AB	Multiply A and B
	PUSH B	Push higher order to stack
	MOV B,A	
	MOV A,R0	
	MUL AB	
	INC DPTR	
	MOVX @DPTR,A	Store the lower order of result
	MOV R2,B	
	POP B	
	MOV A,R0	
	MUL AB	
	ADD A,R2	
	INC DPTR	
	MOVX @DPTR,A	

HERE	MOV A,B INC DPTR MOVX @DPTR,A SJMP HERE	Store the higher order of the result
------	--	--------------------------------------

CUBE OF A NUMBER

Description:

The square of a number is determined by multiplying the value by itself for two times. In this program the input is obtained in A – reg and then it is copied to B – reg and r0 - reg. The values are multiplied and stored tin the memory.

Algorithm:

1. Start the program.
2. Set the origin as 4100H.
3. Initialize DPTR
4. Copy the data to A – reg, B- eg , R0 – reg
5. Multiply the data to find the cube
6. Store the result
7. Stop the program

PROGRAM:

Label	Program	Comments
	MOV DPTR,#4500	Load address of data in DPTR
	MOVX A,@DPTR	Move the data to A
	CPL A	Complement A
	INC A	Increment A by 1.
	INC DPTR	Increment DPTR to store the result of 2's complement of A
	MOVX @DPTR, A	
HERE:	SJMP HERE	

Example:

Input:

4500:03

Output:

4501:27

4502:00

Example:

Input

4500 :01

Output

4501 :F2 (Two's complement)

2'S COMPLIMENT OF A NUMBER

Description:

In order to perform 2's complement in 8051, the given value should be stored in A – reg then take one's complement of A and add value one to LSB.

Algorithm:

1. Load address of data in DPTR
2. Move the data to A
3. Complement A
4. Increment A by 1.
5. Increment DPTR to store the result of 2's complement of A
6. Stop

REVIEW QUESTIONS:

1. Explain the instruction MOV DPTR, #4500H.
2. What does the PUSH instruction do?
3. What instruction is used to multiply any two numbers?
4. What is the function of POP instruction?
5. Which instruction is used to increment the value?
6. What does the ORL instruction do?
7. Explain ANL R1,#0F.
8. How do we take two's complement of number? Give example.
9. What does the ORG 4100H mean?
10. Explain the mode 0 operating mode of 8051 serial ports.
11. Explain the mode 2 operating mode of 8051 serial ports.
12. Explain the mode 3 operating mode of 8051 serial ports.
13. What are the pins used for serial communication?
14. What is the use of SBUF register?
15. What are the methods to double the baud rate?

Result:

Thus the program to determine square, cube and 2's compliment of a number are executed successfully.

Ex. No. 15

Date:

SQUARE WAVE GENERATION USING 8051

Objective:

To write an Assembly Language Program (ALP) to generate square waveform using 8051 instruction set.

Description:

Square waves of any frequency (limited by the controller specifications) can be generated using the 8051 timer. The technique is very simple. Write up a delay subroutine with delay equal to half the time period of the square wave. Make any port pin high and call the delay subroutine. After the delay subroutine is finished, make the corresponding port pin low and call the delay subroutine gain. After the subroutine is finished, repeat the cycle again. The result will be a square wave of the desired frequency at the selected port pin.

Steps:

1. Assume Duty Cycle 50%
2. Assume 12MHZ Clock is Connected to Micro-Controller
3. Use Timers
4. Check output in P3.2

Program for 1 KHz Square wave using 8051 timer

```
ORG 0000H
MOVTMOD, #01H
UP: SETB P3.2
LCALL DELAY
CLR P3.2
LCALL DELAY
SJMP UP
DELAY:MOV TH0,#0FEH
MOV TL0,#0CH
CLR TF0
SETB TR0
HERE:JNB TF0,HERE
RET
END
```

Result:

Thus the square waveform has been generated successfully.