SRM VALLIAMMAI ENGINEERING COLLEGE

(An Autonomous Institution)
SRM NAGAR, KATTANKULATHUR – 603 203.

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
DEPARTMENT OF MEDICAL ELECTRONICS

LAB MANUAL

EC3467 MICROPROCESSORS, MICROCONTROLLERS AND INTERFACING LABORATORY

II-YEAR/IV-SEMESTER

ACADEMIC YEAR: 2025 - 2026 (EVEN SEMESTER)

Prepared by

Dr. A. PANDIAN, AP/ECE

Mrs. K. ARTHI, AP/ECE

Dr. K. DURGADEVI, AP/ECE

Ms. V. VENMATHI, AP/MDE

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.

EC3467 MICROPROCESSORS, MICROCONTROLLERS AND INTERFACING LABORATORY **L T P C** **0 0 4 2**

OBJECTIVES:

1. To develop assembly Language program for 8086 Microprocessor.
2. To introduce string manipulation instructions for 8086 Microprocessor.
3. To understand the working of peripheral devices.
4. To enumerate programs to interface memory, I/O's with processor.
5. To develop assembly Language program for 8051 Microcontroller.
6. To explore the Interfacing of stepper motor and temperature sensor with 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.

LIST OF EXPERIMENTS:

Peripherals and Interfacing Experiments using 8086.

5. Interfacing traffic light controller.
6. Interfacing ADC and DAC.
7. Implementing waveform generation.
8. Interfacing key board and LCD.
9. Analyze serial interface and parallel interface.

LIST OF EXPERIMENTS: 8051 Microcontroller based Experiments using kit and MASM.

10. Program basic arithmetic and logical operations.
11. Interfacing stepper motor and temperature sensor.

LIST OF EXPERIMENTS: PIC18 Microcontroller based Experiments using Proteus software.

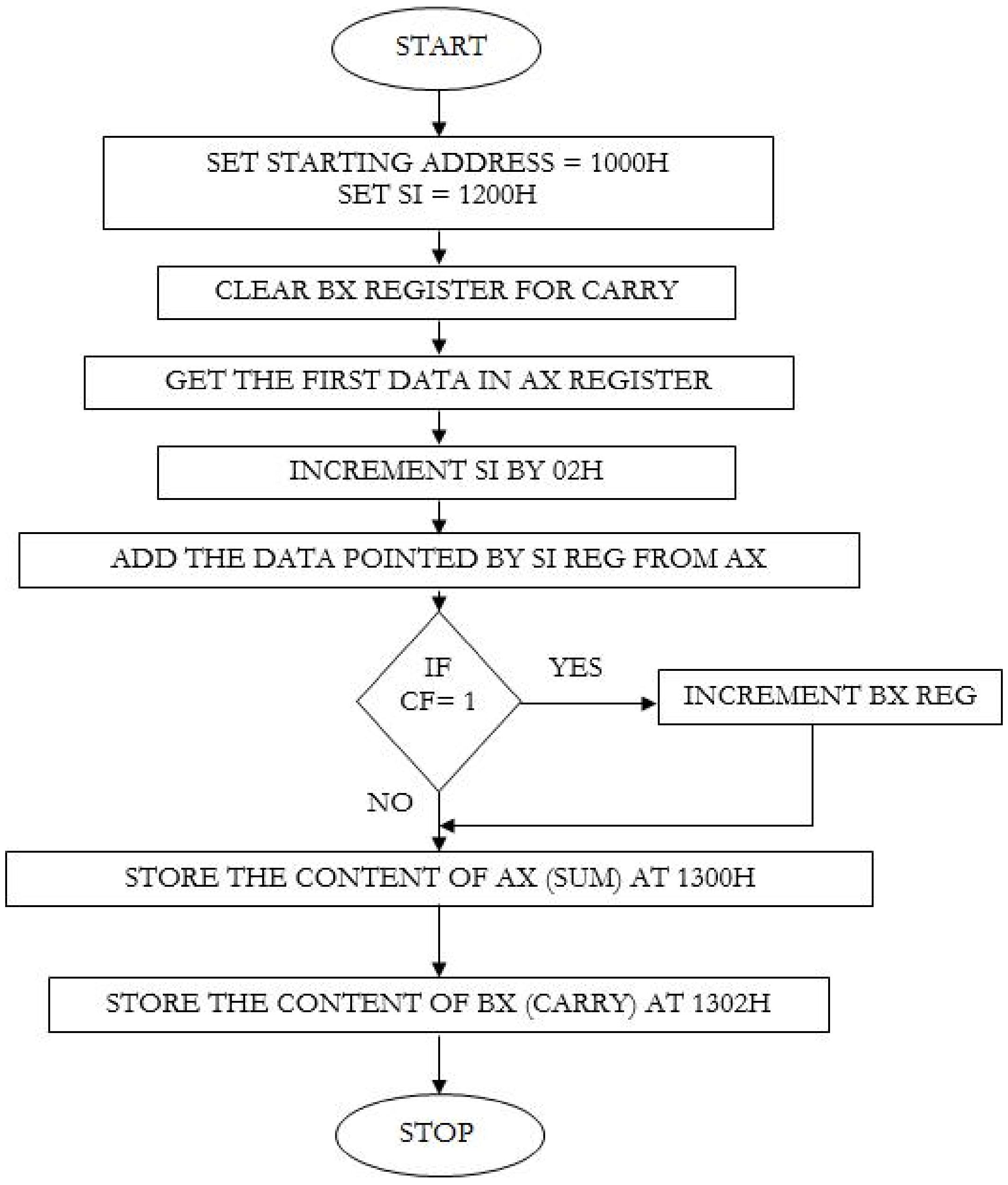
12. Blinking of LED with delay routine
13. Interfacing key board and LCD.
14. Interfacing Temperature sensor

<u>**CONTENTS**</u>

CYCLE I

8086 PROGRAMS & INTERFACING PROGRAMS

Flow Chart for Addition of Two Numbers:


START
SET STARTING ADDRESS = 1000H
SET SI = 1200H
CLEAR BX REGISTER FOR CARRY
GET THE FIRST DATA IN AX REGISTER
INCREMENT SI BY 02H
ADD THE DATA POINTED BY SI REG FROM AX
IF CF= 1
YES
INCREMENT BX REG
NO
STORE THE CONTENT OF AX (SUM) AT 1300H
STORE THE CONTENT OF BX (CARRY) AT 1302H
STOP

Ex. No. 1
Date:

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:

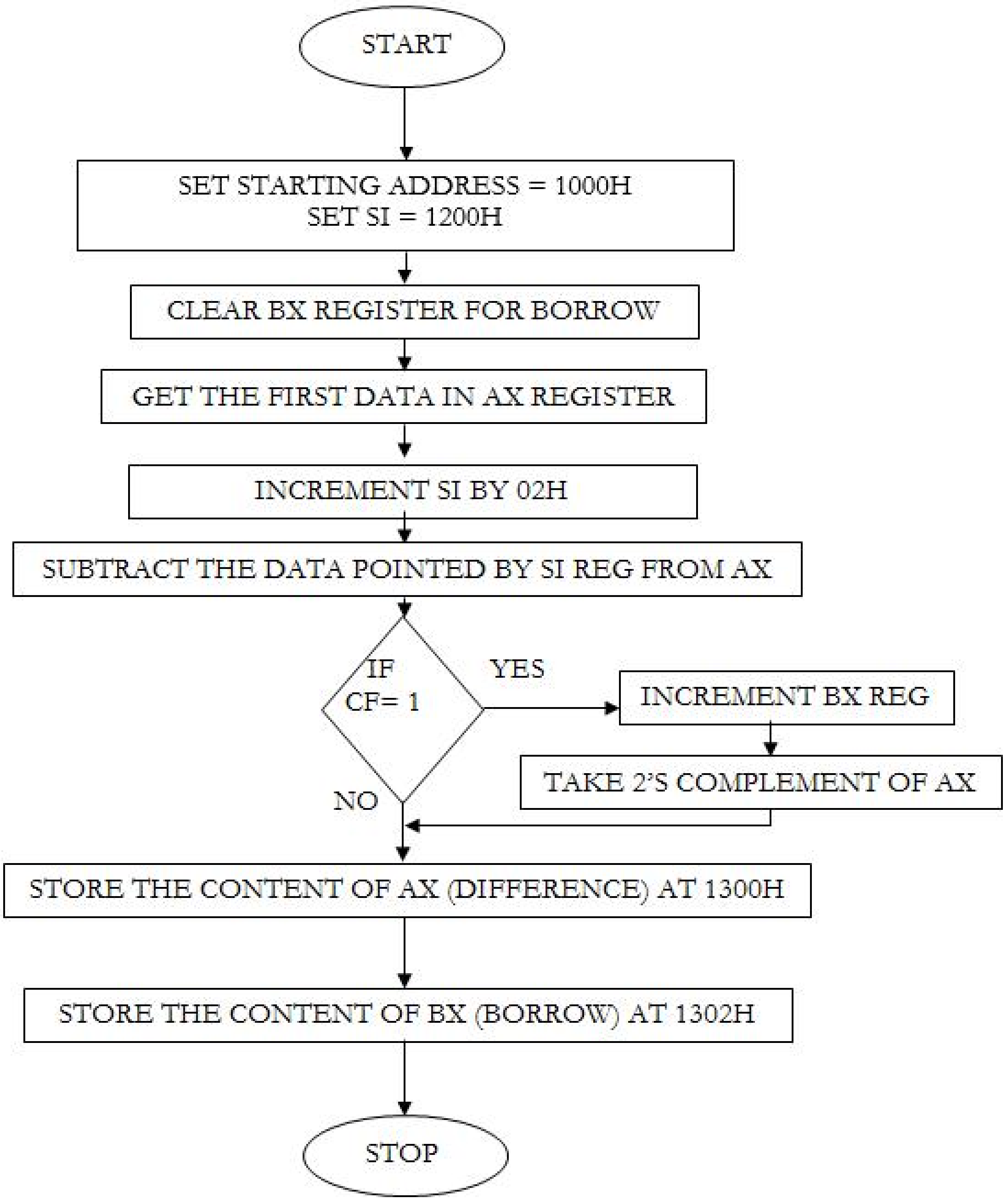
1200:	34H	
1201:	44H	[Addend]
1202:	24H	
1203:	24H	[Augend]

Output:

1300:	58H	
1301:	68H	[Sum]

1302: 00H
1303: 00H [Carry]

Flow Chart of Subtraction of Two Numbers:


START
SET STARTING ADDRESS = 1000H
SET SI = 1200H
CLEAR BX REGISTER FOR BORROW
GET THE FIRST DATA IN AX REGISTER
INCREMENT SI BY 02H
SUBTRACT THE DATA POINTED BY SI REG FROM AX
IF CF= 1
YES
INCREMENT BX REG
TAKE 2'S COMPLEMENT OF AX
NO
STORE THE CONTENT OF AX (DIFFERENCE) AT 1300H
STORE THE CONTENT OF BX (BORROW) AT 1302H
STOP

(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: **Manual Calculation:**

With Borrow

Input:

1200: 03H

1201: 00H (minuend)

1202: 05H
1203: 00H (subtrahend)

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

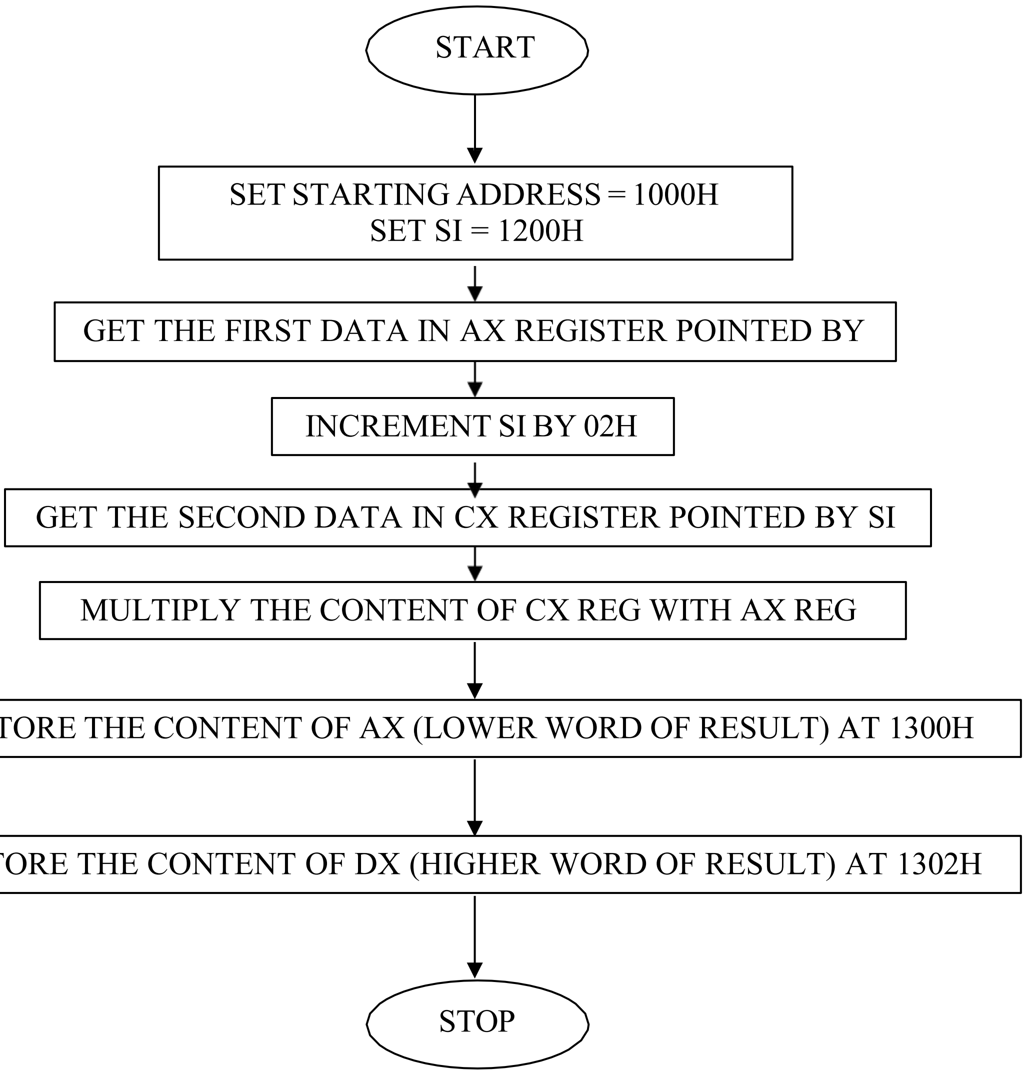
Example 2:

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:


START
SET STARTING ADDRESS = 1000H
SET SI = 1200H
GET THE FIRST DATA IN AX REGISTER POINTED BY
INCREMENT SI BY 02H
GET THE SECOND DATA IN CX REGISTER POINTED BY SI
MULTIPLY THE CONTENT OF CX REG WITH AX REG
STORE THE CONTENT OF AX (LOWER WORD OF RESULT) AT 1300H
STORE THE CONTENT OF DX (HIGHER WORD OF RESULT) AT 1302H
STOP

(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:

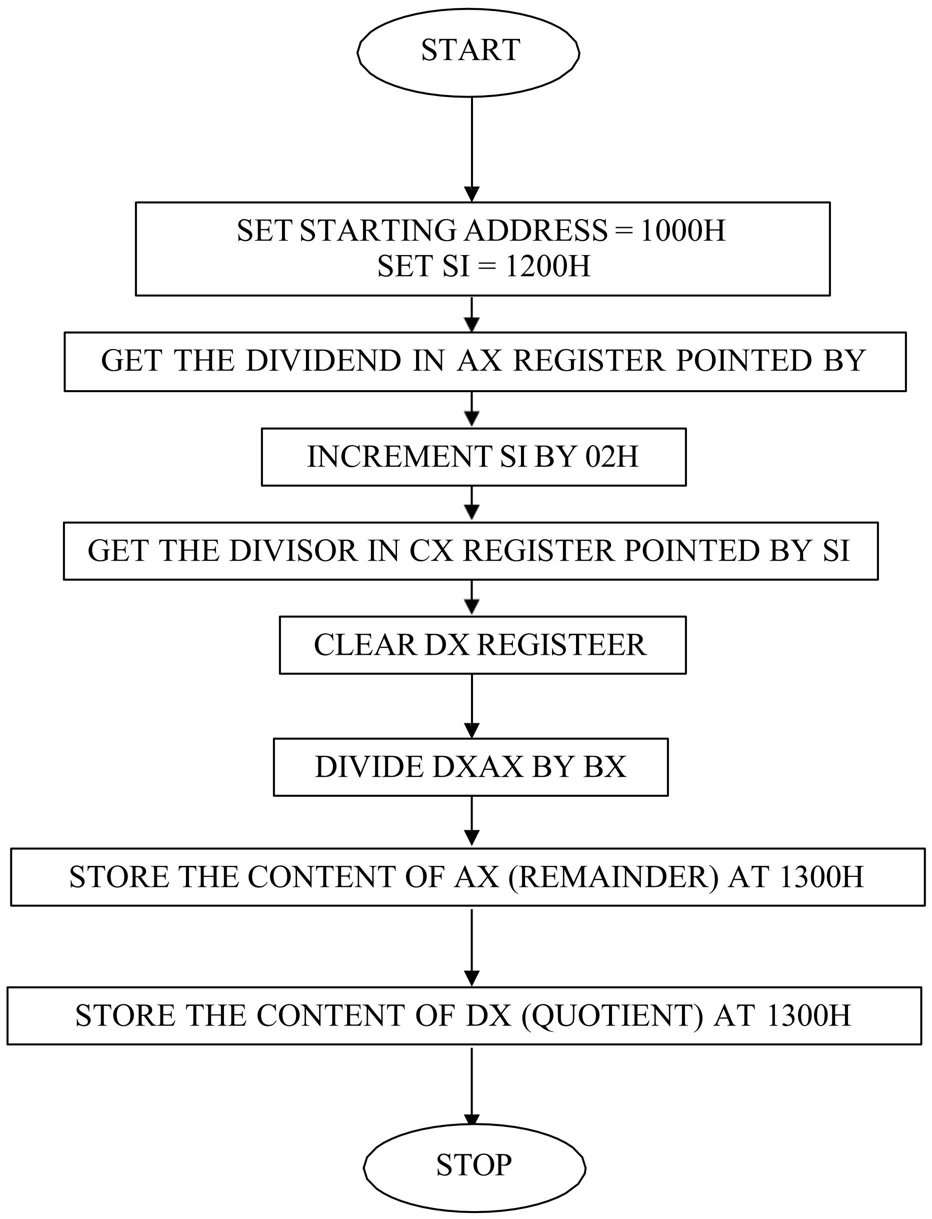
1200: 02H
1201: 06H (Multiplicand)
1202: 02H

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:


START
SET STARTING ADDRESS = 1000H
SET SI = 1200H
GET THE DIVIDEND IN AX REGISTER POINTED BY
INCREMENT SI BY 02H
GET THE DIVISOR IN CX REGISTER POINTED BY SI
CLEAR DX REGISTEER
DIVIDE DXAX BY BX
STORE THE CONTENT OF AX (REMAINDER) AT 1300H
STORE THE CONTENT OF DX (QUOTIENT) AT 1300H
STOP

(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:

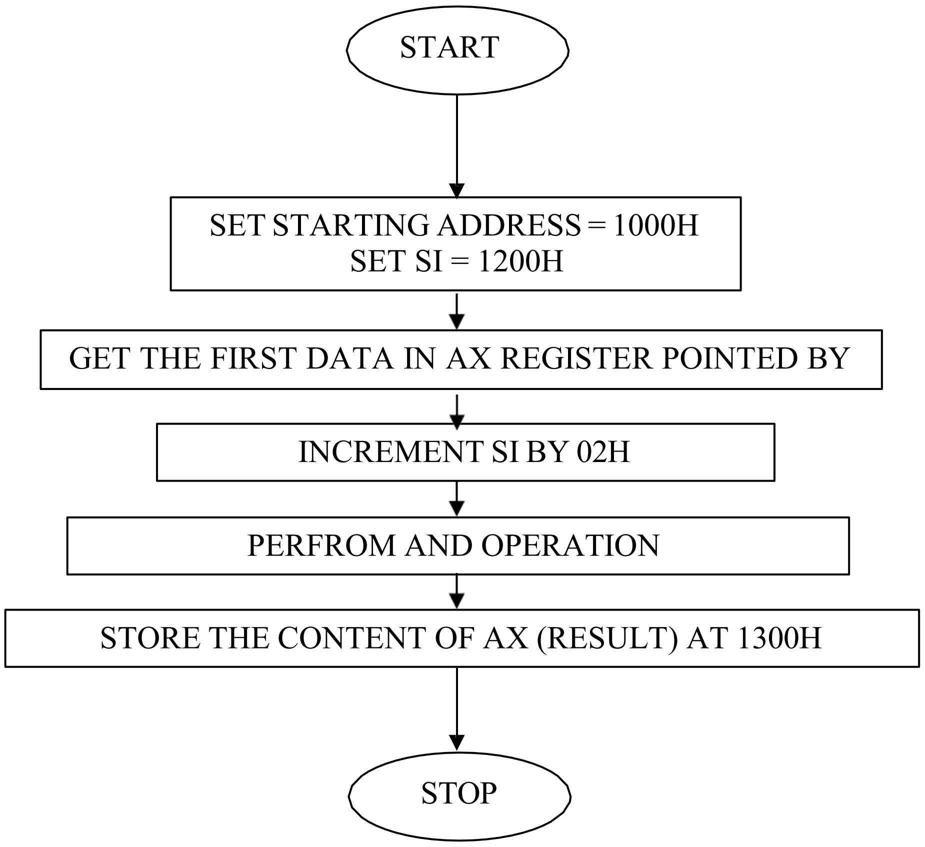
```
1200:   06H
1201:   06H    (Dividend)
1202:   03H
1203:   03H    (Divisor)
```

Output:

```
1300:   02H
1301:   00H    (Quotient)
```

1302: 00H
1303: 00H (Remainder)

FLOWCHART


START
SET STARTING ADDRESS = 1000H
SET SI = 1200H
GET THE FIRST DATA IN AX REGISTER POINTED BY
INCREMENT SI BY 02H
PERFROM AND OPERATION
STORE THE CONTENT OF AX (RESULT) AT 1300H
STOP

(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	Set starting address as 1000H.
	MOV SI,1200H	Initialize SI
	MOV AX,[SI]	Get the first data in AX – reg
	ADD SI,02H	Increment SI to point next data
	AND AX,[SI]	Perform AND operation of two data

	MOV DI,1300H MOV [DI],AX HLT	Store the result in memory

Example: **Manual Calculation:**

Input

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

Output

1300:00H
1301:00H

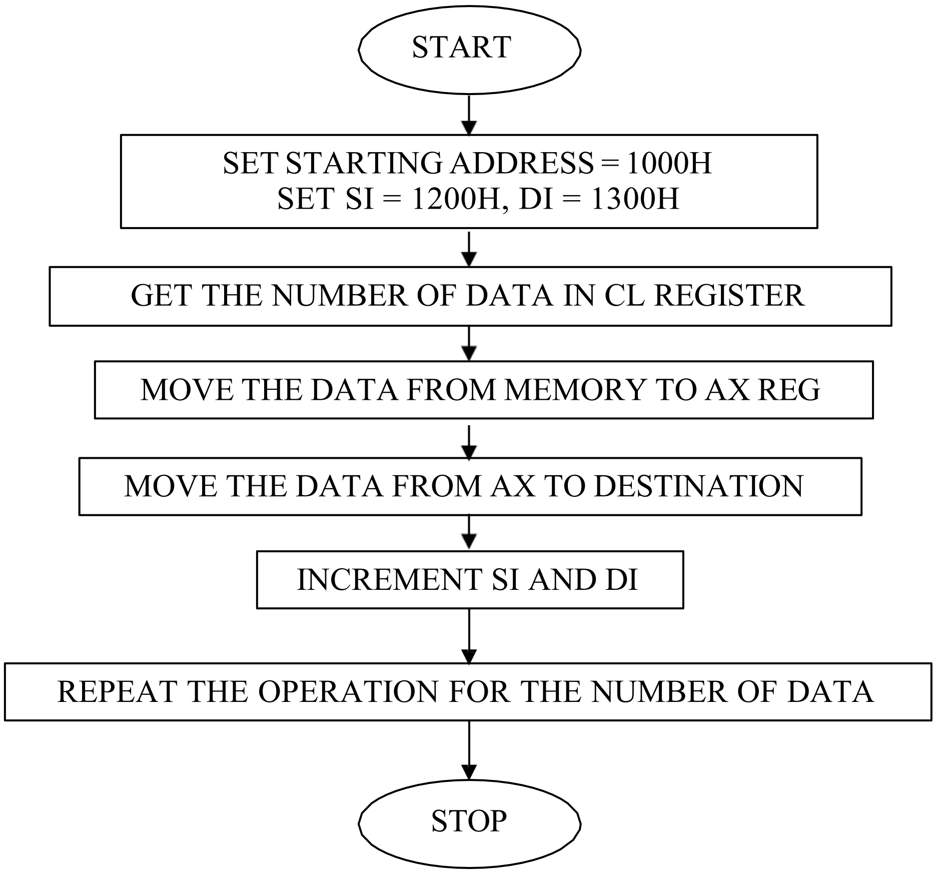
REVIEW QUESTIONS:

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:


START
SET STARTING ADDRESS = 1000H
SET SI = 1200H, DI = 1300H
GET THE NUMBER OF DATA IN CL REGISTER
MOVE THE DATA FROM MEMORY TO AX REG
MOVE THE DATA FROM AX TO DESTINATION
INCREMENT SI AND DI
REPEAT THE OPERATION FOR THE NUMBER OF DATA
STOP

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

Example:

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

Manual Calculation:

Output:

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

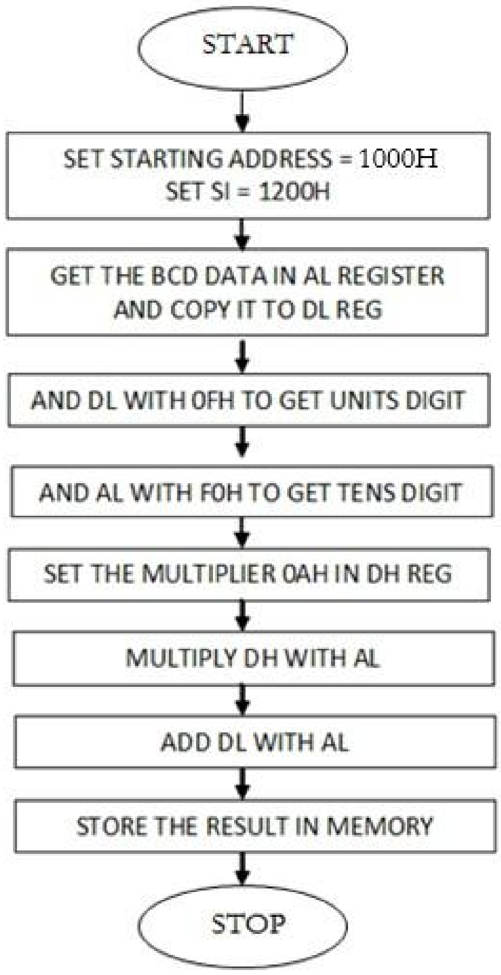
REVIEW QUESTIONS:

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?

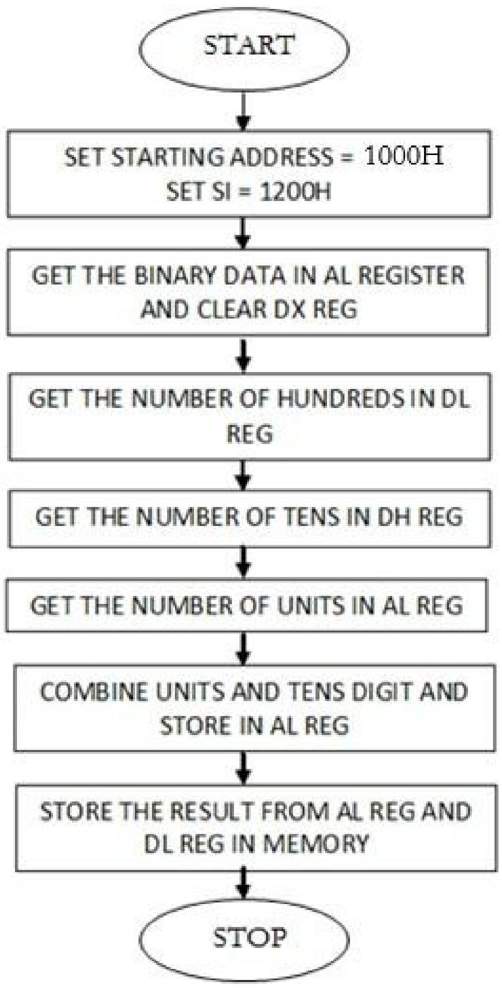
Result:

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

Flow Chart for Binary to BCD Conversion


START
SET STARTING ADDRESS = 1000H
SET SI = 1200H
GET THE BCD DATA IN AL REGISTER AND COPY IT TO DL REG
AND DL WITH 0FH TO GET UNITS DIGIT
AND AL WITH F0H TO GET TENS DIGIT
SET THE MULTIPLIER 0AH IN DH REG
MULTIPLY DH WITH AL
ADD DL WITH AL
STORE THE RESULT IN MEMORY
STOP

Flow Chart for BCD to Binary Conversion


START
SET STARTING ADDRESS = 1000H
SET SI = 1200H
GET THE BINARY DATA IN AL REGISTER AND CLEAR DX REG
GET THE NUMBER OF HUNDREDS IN DL REG
GET THE NUMBER OF TENS IN DH REG
GET THE NUMBER OF UNITS IN AL REG
COMBINE UNITS AND TENS DIGIT AND STORE IN AL REG
STORE THE RESULT FROM AL REG AND DL REG IN MEMORY
STOP

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:

Manual Calculation:

Input:

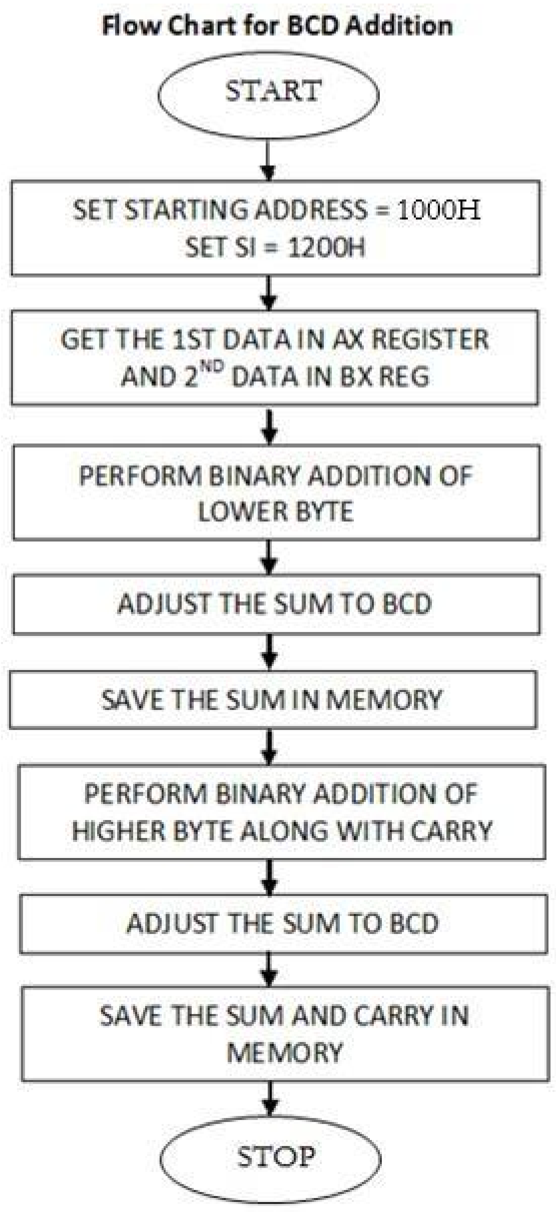
1200: 55H [Binary data]

Output:

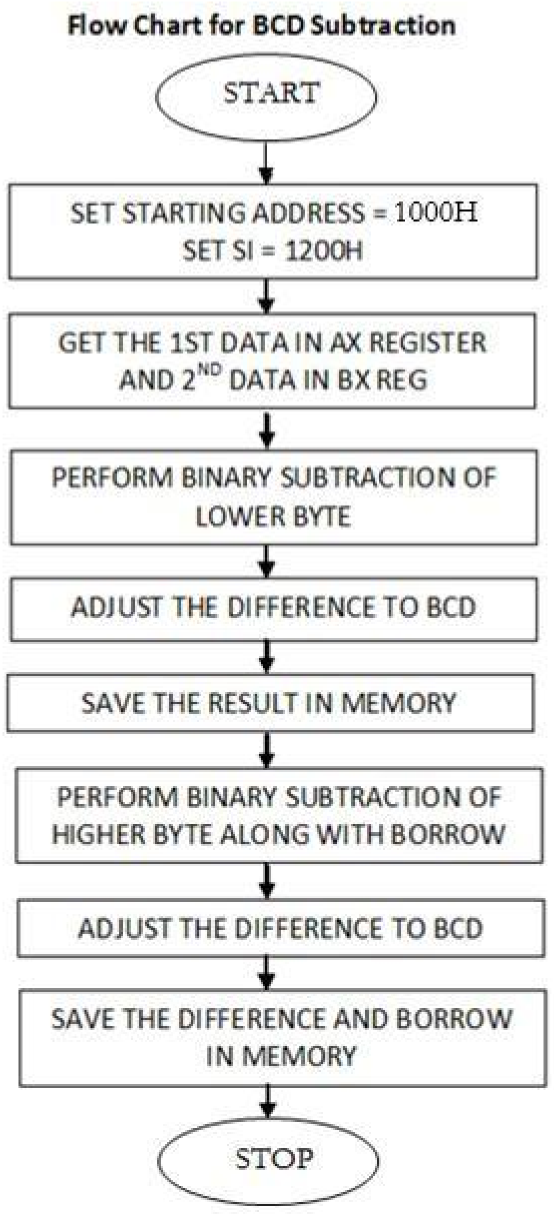
1201:85H

Result:

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



Flow Chart for BCD Addition
START
SET STARTING ADDRESS = 1000H
SET SI = 1200H
GET THE 1ST DATA IN AX REGISTER AND 2ND DATA IN BX REG
PERFORM BINARY ADDITION OF LOWER BYTE
ADJUST THE SUM TO BCD
SAVE THE SUM IN MEMORY
PERFORM BINARY ADDITION OF HIGHER BYTE ALONG WITH CARRY
ADJUST THE SUM TO BCD
SAVE THE SUM AND CARRY IN MEMORY
STOP



Flow Chart for BCD Subtraction
START
SET STARTING ADDRESS = 1000H
SET SI = 1200H
GET THE 1ST DATA IN AX REGISTER AND 2ND DATA IN BX REG
PERFORM BINARY SUBTRACTION OF LOWER BYTE
ADJUST THE DIFFERENCE TO BCD
SAVE THE RESULT IN MEMORY
PERFORM BINARY SUBTRACTION OF HIGHER BYTE ALONG WITH BORROW
ADJUST THE DIFFERENCE TO BCD
SAVE THE DIFFERENCE AND BORROW IN MEMORY
STOP

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 1st number in AX reg
	MOV BX,[SI+2]	Get the 2nd 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 1st number in AX reg
	MOV BX,[SI+2]	Get the 2nd 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]

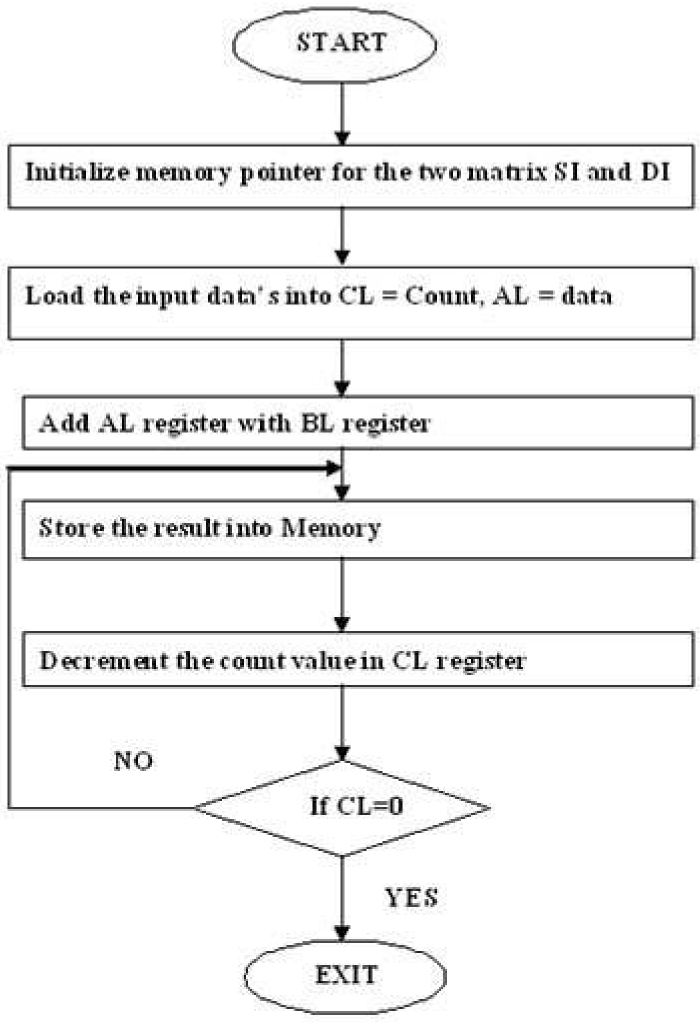
Output:

1204: 09
1205: 02
1203: 02

Result:

Thus the program for BCD subtraction was successfully executed.

Flow Chart for Matrix Addition:


START
Initialize memory pointer for the two matrix SI and DI
Load the input data's into CL = Count, AL = data
Add AL register with BL register
Store the result into Memory
Decrement the count value in CL register
NO
If CL=0
YES
EXIT

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 1st matrix
	MOV DI, 3000	Load 3000 into DI for 2nd 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:

Manual Calculation:

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

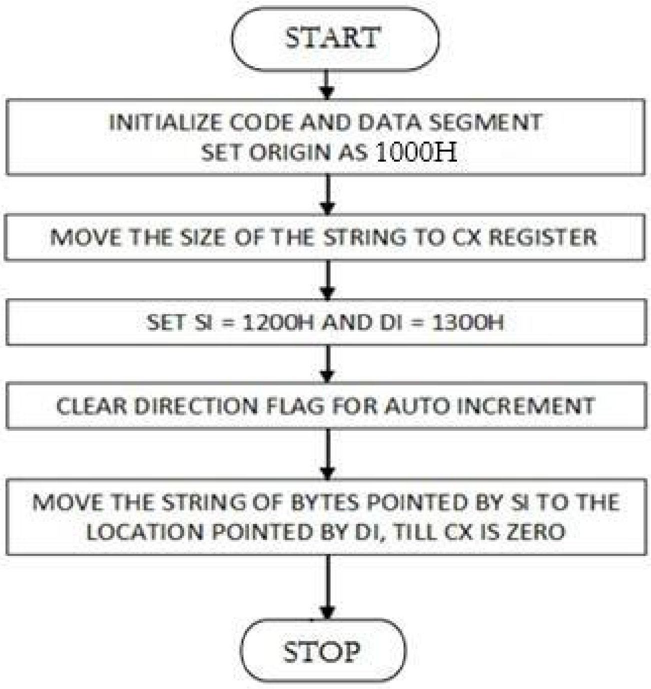
REVIEW QUESTIONS:

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?

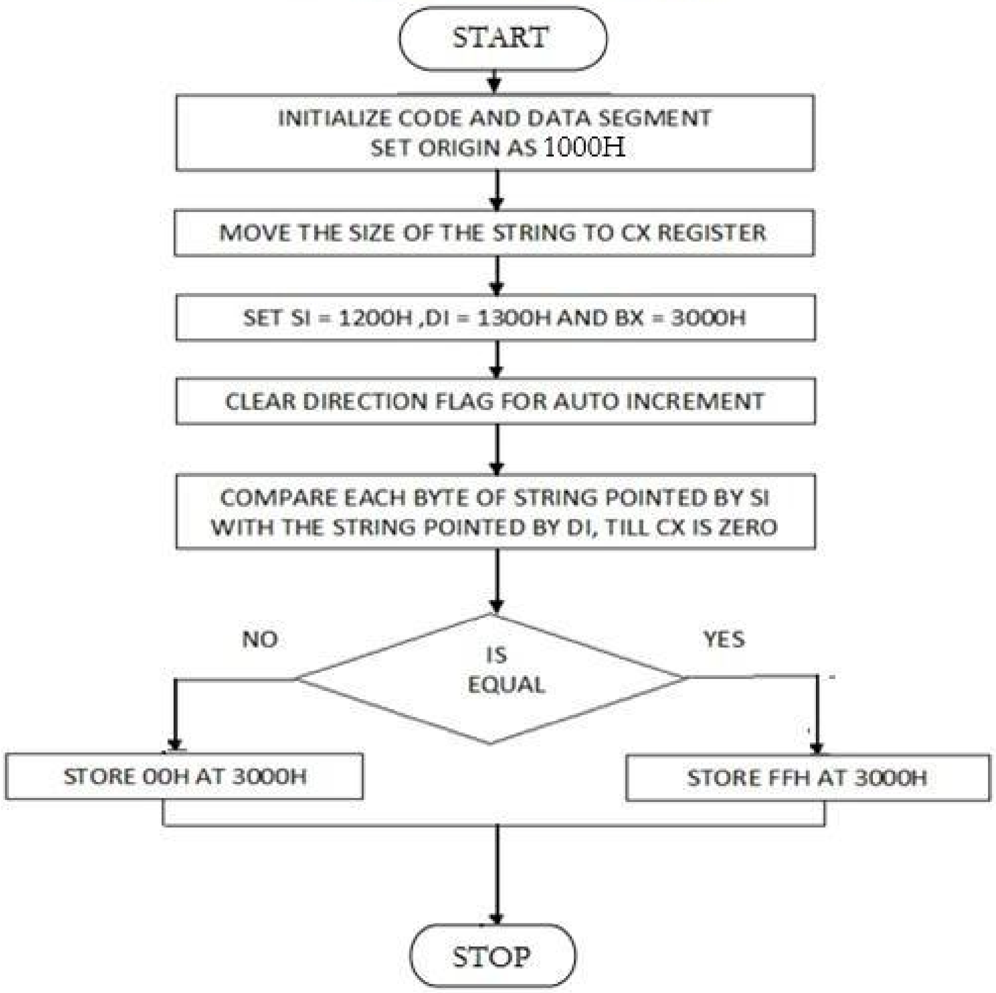
Result:

Thus the program for Matrix multiplication was successfully executed.

Flowchart for Copying a String


START
INITIALIZE CODE AND DATA SEGMENT
SET ORIGIN AS 1000H
MOVE THE SIZE OF THE STRING TO CX REGISTER
SET SI = 1200H AND DI = 1300H
CLEAR DIRECTION FLAG FOR AUTO INCREMENT
MOVE THE STRING OF BYTES POINTED BY SI TO THE LOCATION POINTED BY DI, TILL CX IS ZERO
STOP

Flowchart for Comparing two Strings


START
INITIALIZE CODE AND DATA SEGMENT
SET ORIGIN AS 1000H
MOVE THE SIZE OF THE STRING TO CX REGISTER
SET SI = 1200H ,DI = 1300H AND BX = 3000H
CLEAR DIRECTION FLAG FOR AUTO INCREMENT
COMPARE EACH BYTE OF STRING POINTED BY SI WITH THE STRING POINTED BY DI, TILL CX IS ZERO
NO
IS EQUAL
YES
STORE 00H AT 3000H
STORE FFH AT 3000H
STOP

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:

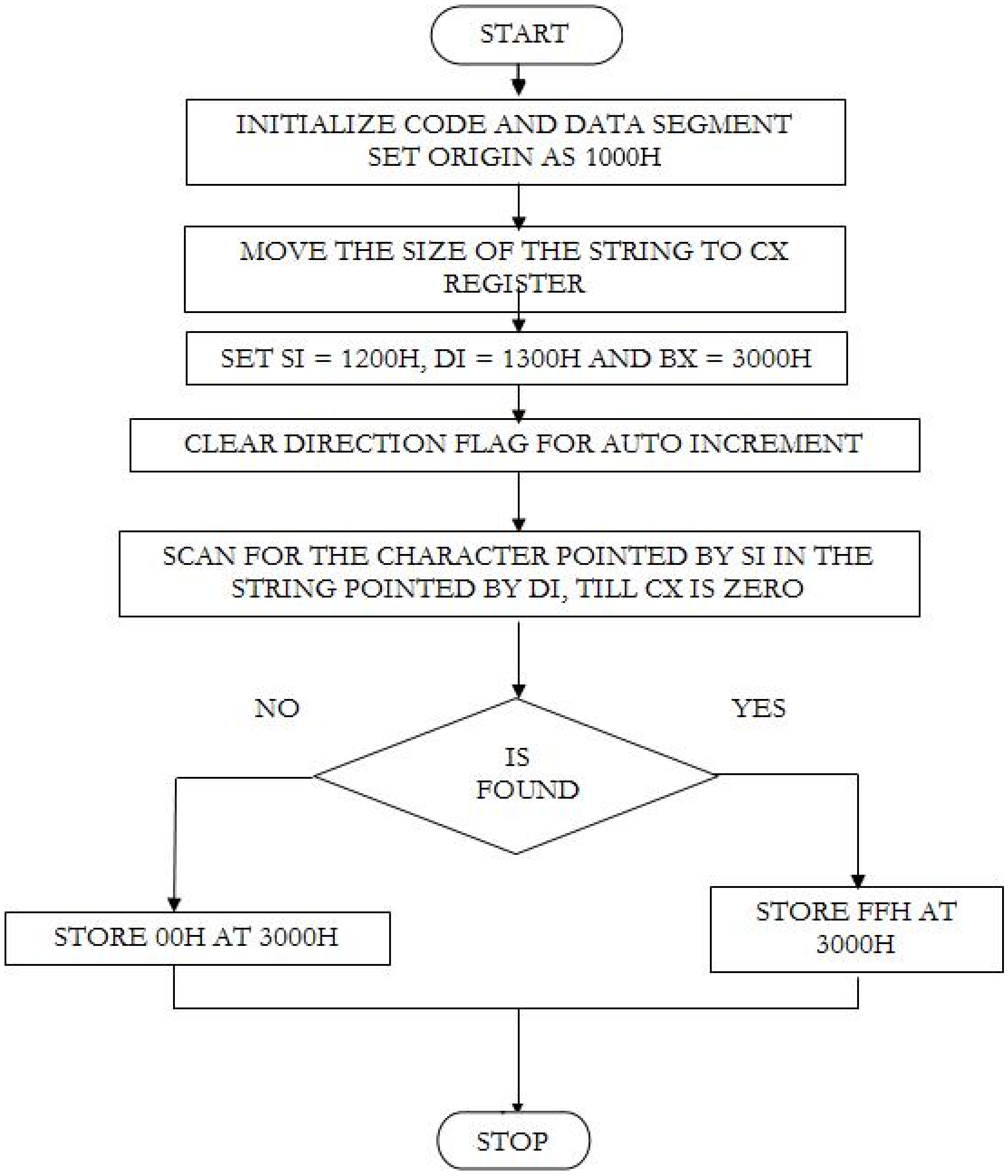
```
1200:   02
1201:   03
1202:   04
1203:   05
1204:   06

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

Output:

```
3000:   00H
```

Manual Calculation:

Flow Chart for Scan a Character in a String:


START
INITIALIZE CODE AND DATA SEGMENT
SET ORIGIN AS 1000H
MOVE THE SIZE OF THE STRING TO CX REGISTER
SET SI = 1200H, DI = 1300H AND BX = 3000H
CLEAR DIRECTION FLAG FOR AUTO INCREMENT
SCAN FOR THE CHARACTER POINTED BY SI IN THE STRING POINTED BY DI, TILL CX IS ZERO
NO
YES
IS FOUND
STORE 00H AT 3000H
STORE FFH AT 3000H
STOP

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:

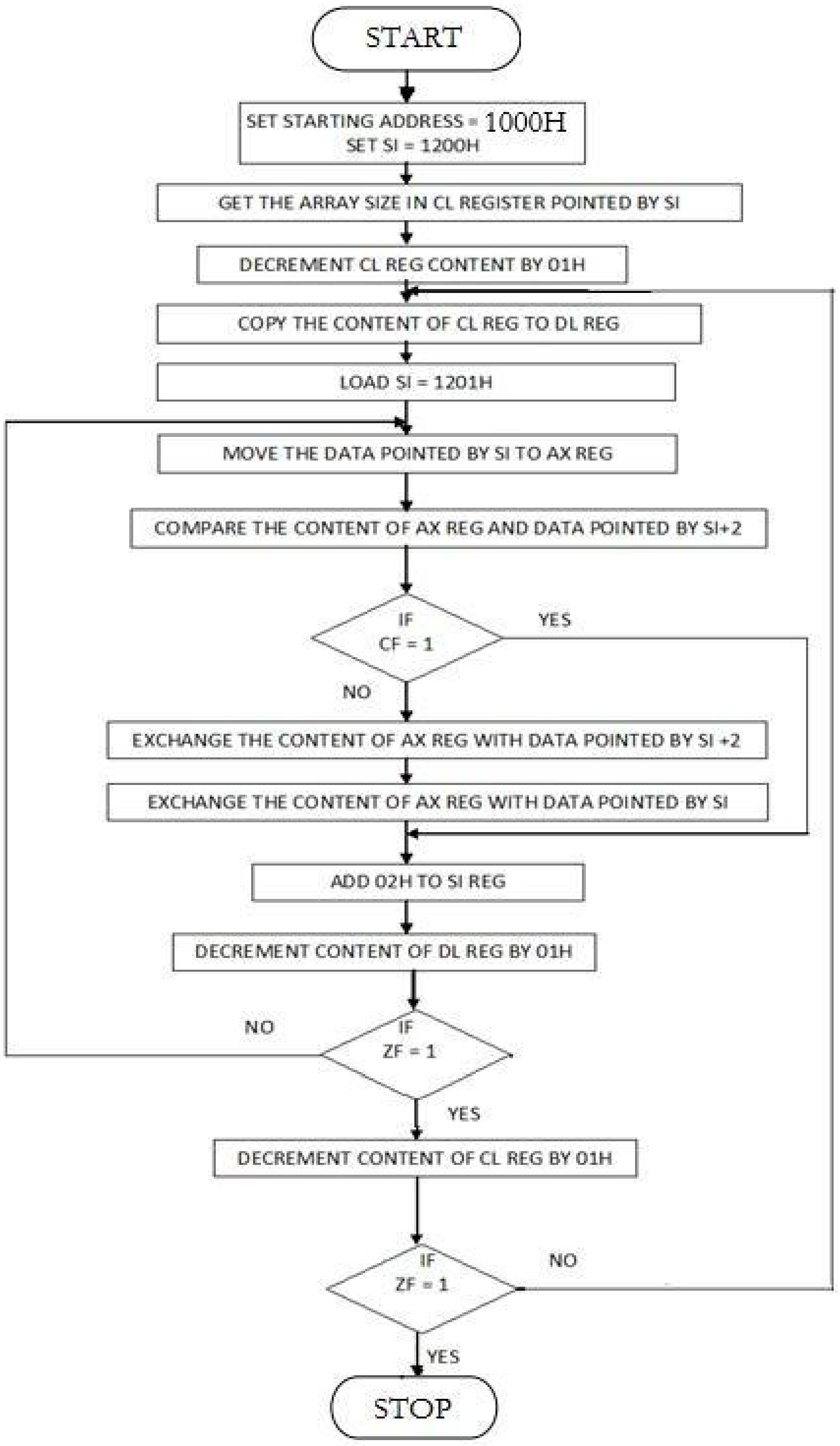
Label	**Program**	**Comments**
	ORG 1000H	Set the starting address as 1000H.
	MOV CX, 0004H	Move immediate data to CX.
	MOV SI, 1200H	Move immediate data to SI.
	MOV AL, [SI]	Move contents of SI to AL.
	MOV DI, 1300H	Move immediate data to DI.
	MOV BX, 3000H	Move immediate data to BX.
	CLD	Clear directional 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:	**Input:**
1200:AD (Data to be scanned)	1200: BB (Data to be scanned)
1300:AA	1300:AA
1301:AB	1301:AB
1302:AA	1302:AA
1303:AD	1303:AD
Output:	**Output:**
3000:FF	3000:00

Manual Calculation:

Flow Chart for Sorting – Ascending Order


START
SET STARTING ADDRESS = 1000H
SET SI = 1200H
GET THE ARRAY SIZE IN CL REGISTER POINTED BY SI
DECREMENT CL REG CONTENT BY 01H
COPY THE CONTENT OF CL REG TO DL REG
LOAD SI = 1201H
MOVE THE DATA POINTED BY SI TO AX REG
COMPARE THE CONTENT OF AX REG AND DATA POINTED BY SI+2
IF CF = 1
YES
NO
EXCHANGE THE CONTENT OF AX REG WITH DATA POINTED BY SI +2
EXCHANGE THE CONTENT OF AX REG WITH DATA POINTED BY SI
ADD 02H TO SI REG
DECREMENT CONTENT OF DL REG BY 01H
NO
IF ZF = 1
YES
DECREMENT CONTENT OF CL REG BY 01H
IF ZF = 1
NO
YES
STOP

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 toe 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

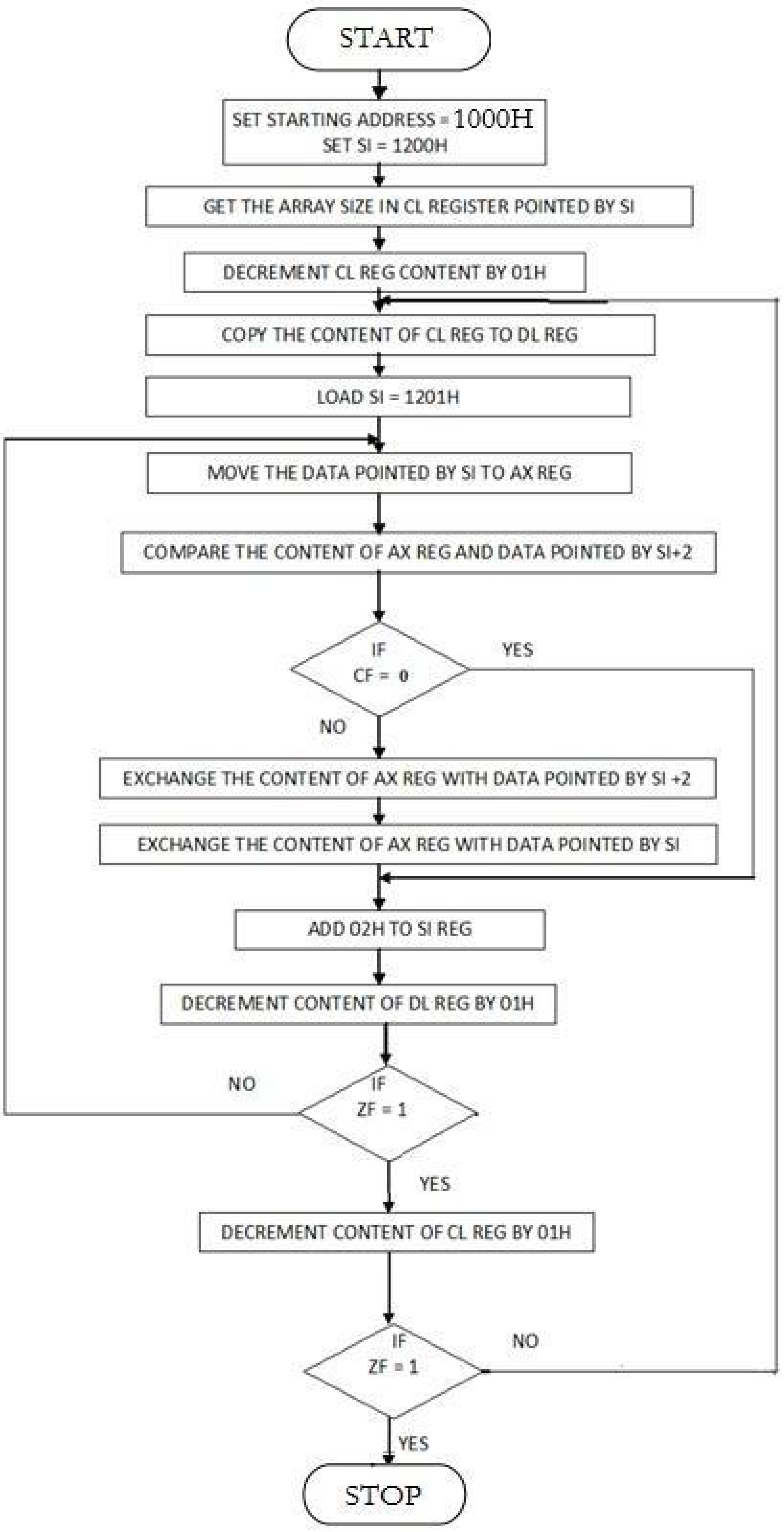
PROGRAM:

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

Example:

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

Flow Chart for Sorting - Descending Order


START
SET STARTING ADDRESS = 1000H
SET SI = 1200H
GET THE ARRAY SIZE IN CL REGISTER POINTED BY SI
DECREMENT CL REG CONTENT BY 01H
COPY THE CONTENT OF CL REG TO DL REG
LOAD SI = 1201H
MOVE THE DATA POINTED BY SI TO AX REG
COMPARE THE CONTENT OF AX REG AND DATA POINTED BY SI+2
IF CF = 0
YES
NO
EXCHANGE THE CONTENT OF AX REG WITH DATA POINTED BY SI +2
EXCHANGE THE CONTENT OF AX REG WITH DATA POINTED BY SI
ADD 02H TO SI REG
DECREMENT CONTENT OF DL REG BY 01H
IF ZF = 1
NO
YES
DECREMENT CONTENT OF CL REG BY 01H
IF ZF = 1
NO
YES
STOP

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

Example:

Input:

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

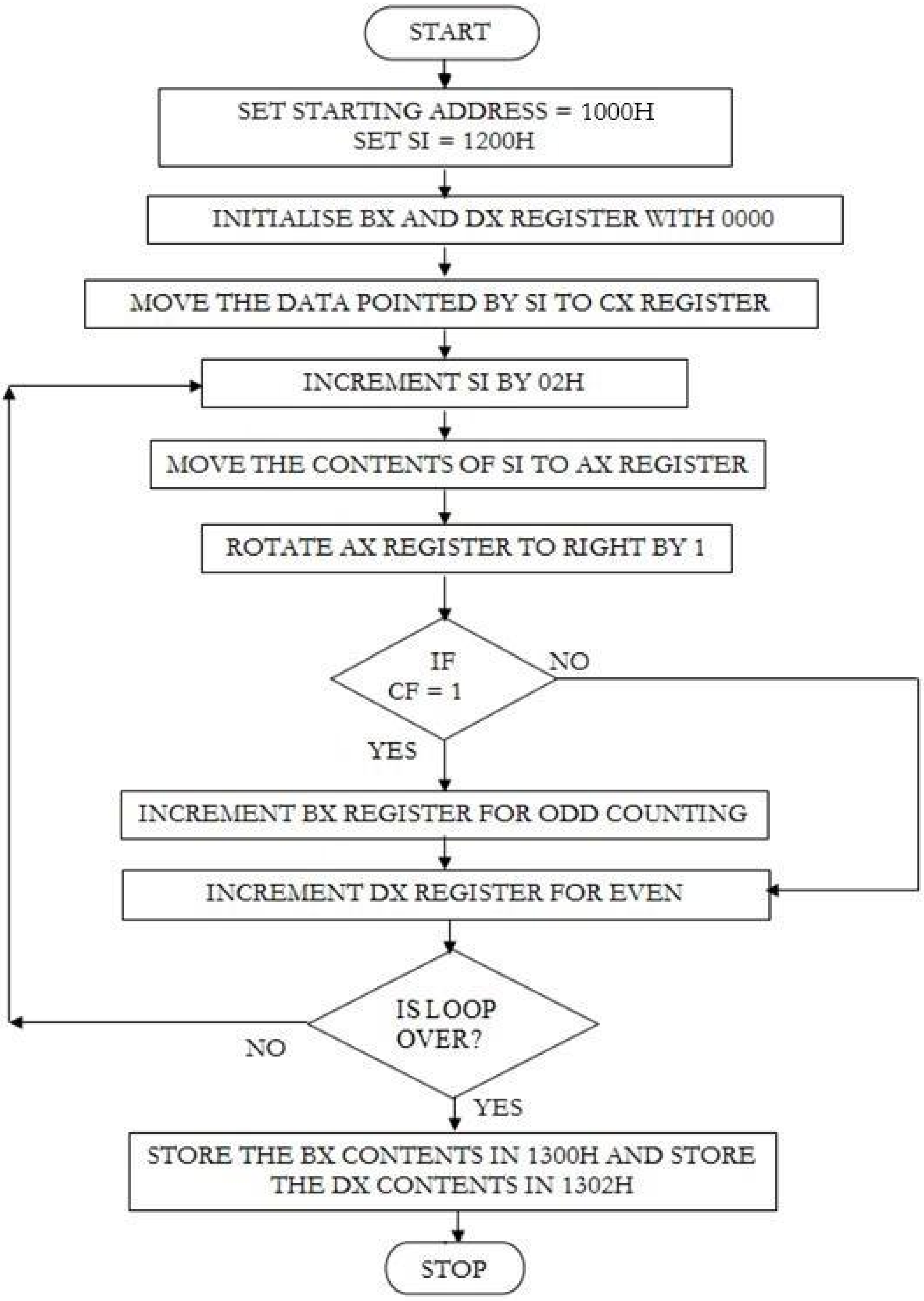
1208:38

Output:

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

Manual Calculation:

Flow Chart for Searching Odd-Even Numbers:



START
SET STARTING ADDRESS = 1000H
SET SI = 1200H
INITIALISE BX AND DX REGISTER WITH 0000
MOVE THE DATA POINTED BY SI TO CX REGISTER
INCREMENT SI BY 02H
MOVE THE CONTENTS OF SI TO AX REGISTER
ROTATE AX REGISTER TO RIGHT BY 1
IF CF = 1
NO
YES
INCREMENT BX REGISTER FOR ODD COUNTING
INCREMENT DX REGISTER FOR EVEN
IS LOOP OVER?
NO
YES
STORE THE BX CONTENTS IN 1300H AND STORE THE DX CONTENTS IN 1302H
STOP

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

Example:

Input: **Manual Calculation:**

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

Output:

```
1300:01   odd
1301:03    even
```

<u>**REVIEW QUESTIONS:**</u>

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:

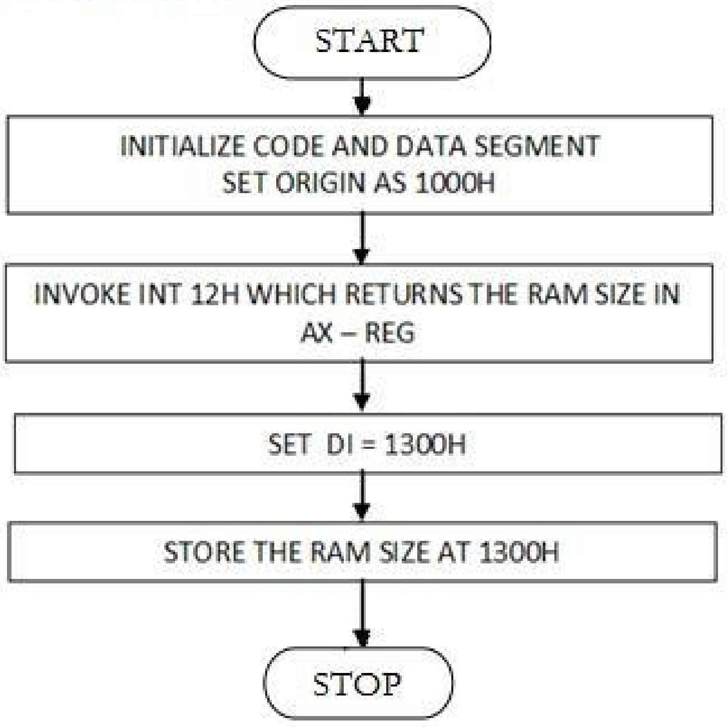
Label	Program	Comments
	DATA SEGMENT PASSWORD DB 'MASM1234' LEN EQU ($-PASSWORD) MSG1 DB 10,13,'ENTER YOUR PASSWORD: $' MSG2 DB 10,13,'YOUR PASSWORD IS CORRECT!!$' MSG3 DB 10,13,'INCORRECT PASSWORD!$' NEW DB 10,13,'$' INST DB 10 DUP(0) DATA ENDS CODE SEGMENT ASSUME CS:CODE,DS:DATA ORG 1000H START: MOV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

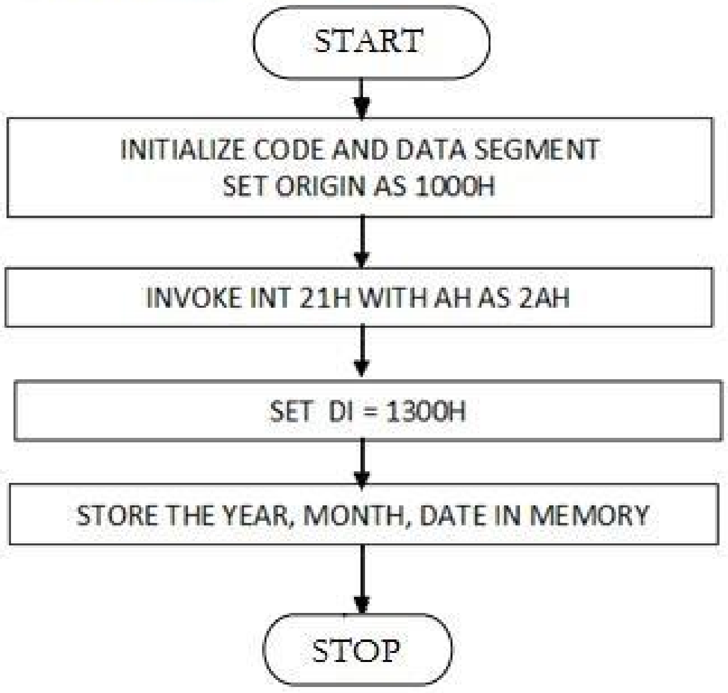
Label	Program	Comments
	MOV [INST+SI],AL MOV DL,'*' MOV AH,02H INT 21H INC SI JMP UP1 DOWN: MOV BX,00 MOV CX,LEN CHECK: MOV AL,[INST+BX] MOV DL,[PASSWORD+BX] CMP AL,DL JNE FAIL INC BX LOOP CHECK LEA DX,MSG2 MOV AH,09H INT 21H JMP FINISH FAIL: LEA DX,MSG3 MOV AH,009H INT 21H FINISH: INT 3 CODE ENDS END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

Observation:

Flow Chart to Print RAM Size


START
INITIALIZE CODE AND DATA SEGMENT
SET ORIGIN AS 1000H
INVOKE INT 12H WHICH RETURNS THE RAM SIZE IN AX – REG
SET DI = 1300H
STORE THE RAM SIZE AT 1300H
STOP

Flow Chart to Print System Date


START
INITIALIZE CODE AND DATA SEGMENT
SET ORIGIN AS 1000H
INVOKE INT 21H WITH AH AS 2AH
SET DI = 1300H
STORE THE YEAR, MONTH, DATE IN MEMORY
STOP

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

Example: **Manual Calculation:**

Output:

1300: 80

Program:

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

Ex. No. 6
Date:

INTERFACING TRAFFIC LIGHT CONTROL

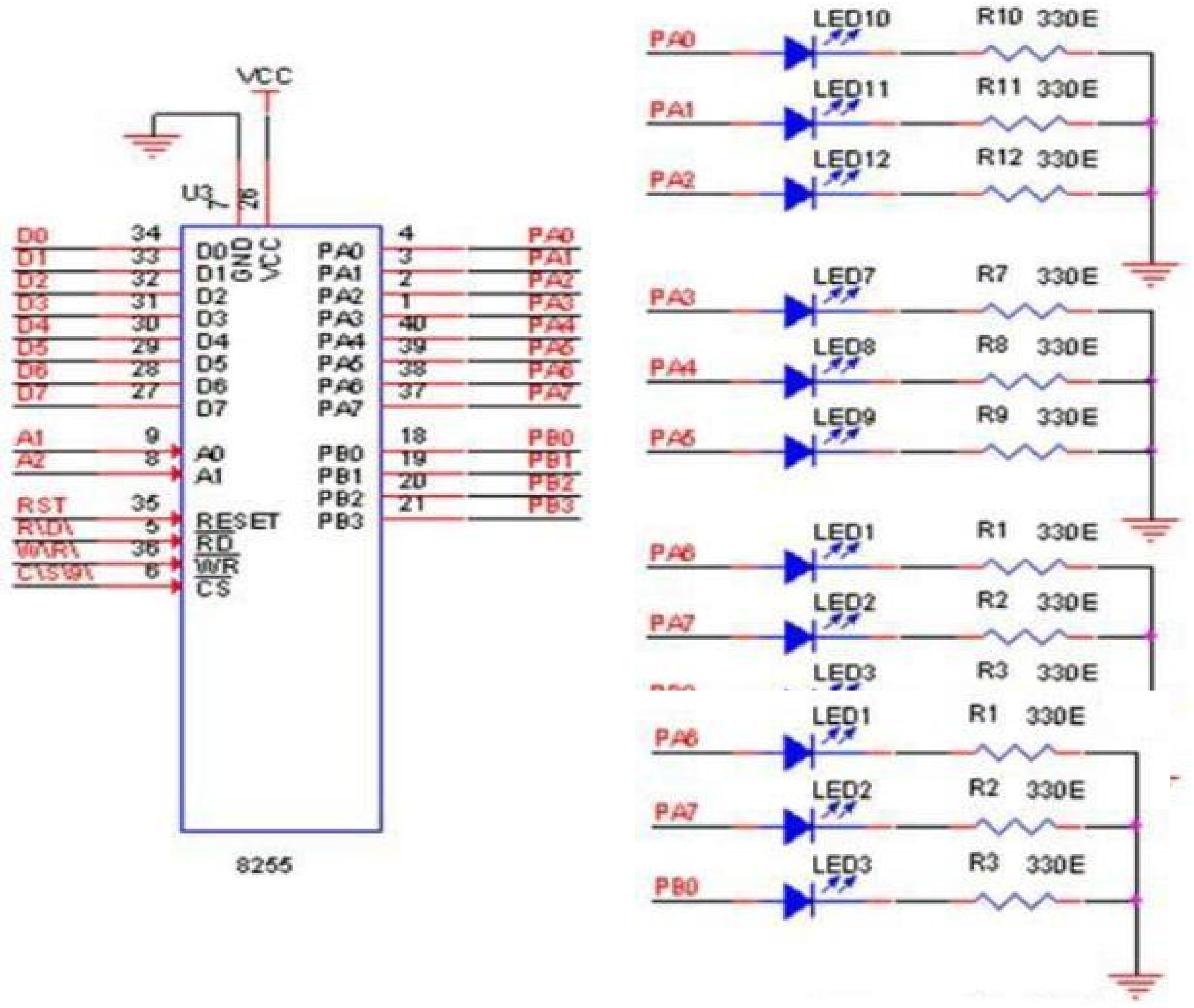
AIM

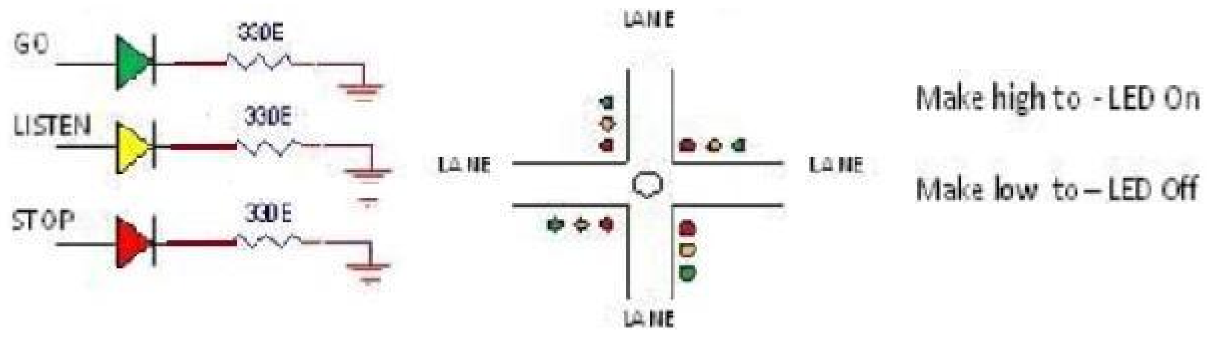
To write an 8086 assembly language program to interface the traffic light controller with 8255 and verify the operation.

DESCRIPTION

The system is a simple contraption 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





PROGRAM:

Label	Mnemonics

START	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	MOV DX, FFFF
AGAIN	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
```

<u>REVIEW QUSETIONS:</u>

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:

ANALOG TO DIGITAL CONVERSION 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 µs 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
1	1	0	1	1	X	X	X

=D8H

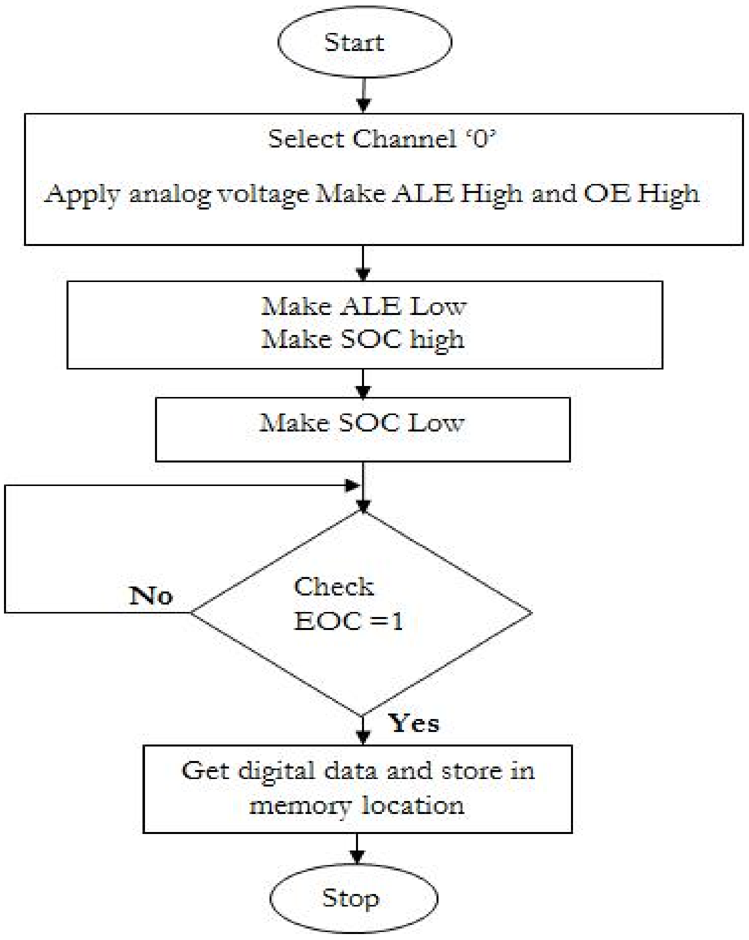
The EOC 2 is selected when

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

=E0H

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

FLOWHCART


Start
Select Channel '0'
Apply analog voltage Make ALE High and OE High
Make ALE Low
Make SOC high
Make SOC Low
Check
EOC =1
No
Yes
Get digital data and store in memory location
Stop

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:

<u>REVIEW QUSETIONS:</u>

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?

Result:

Thus the program to demonstrate the ADC was executed.

Ex. No. 8
Date:

DIGITAL TO ANALOG CONVERSION INTERFACE

Aim:

To write an assembly language program to demonstrate Digital to Analog Conversion

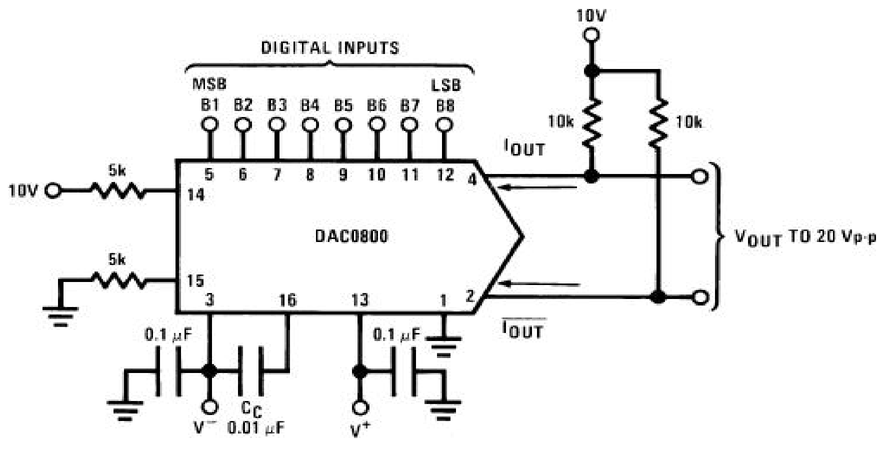
THEORY:

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

DAC 0800

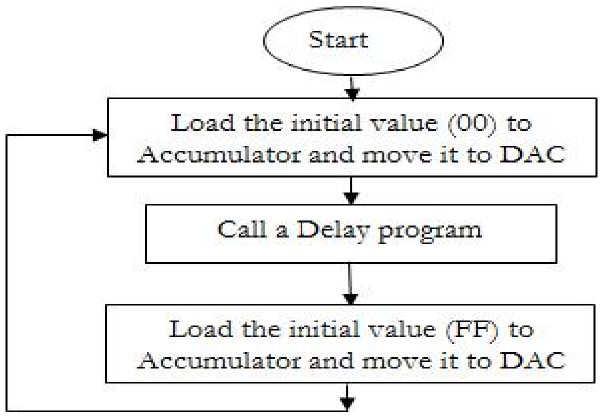


ALGORITHM:

(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


Start
Load the initial value (00) to Accumulator and move it to DAC
Call a Delay program
Load the initial value (FF) to Accumulator and move it to DAC

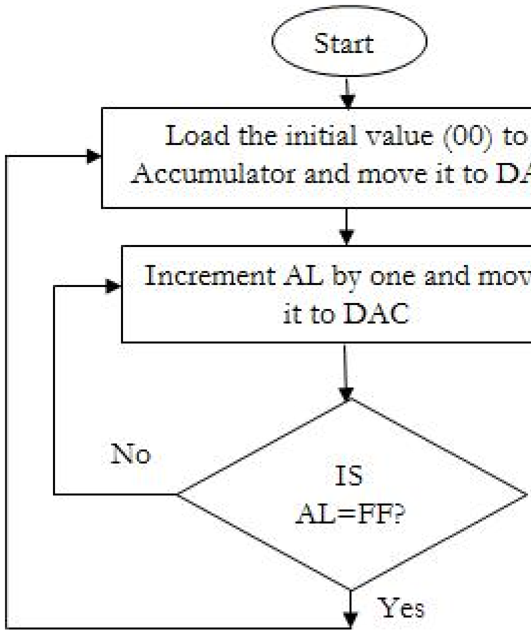
PROGRAM

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

(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


Start
Load the initial value (00) to Accumulator and move it to DAC
Increment AL by one and move it to DAC
IS AL=FF?
No
Yes

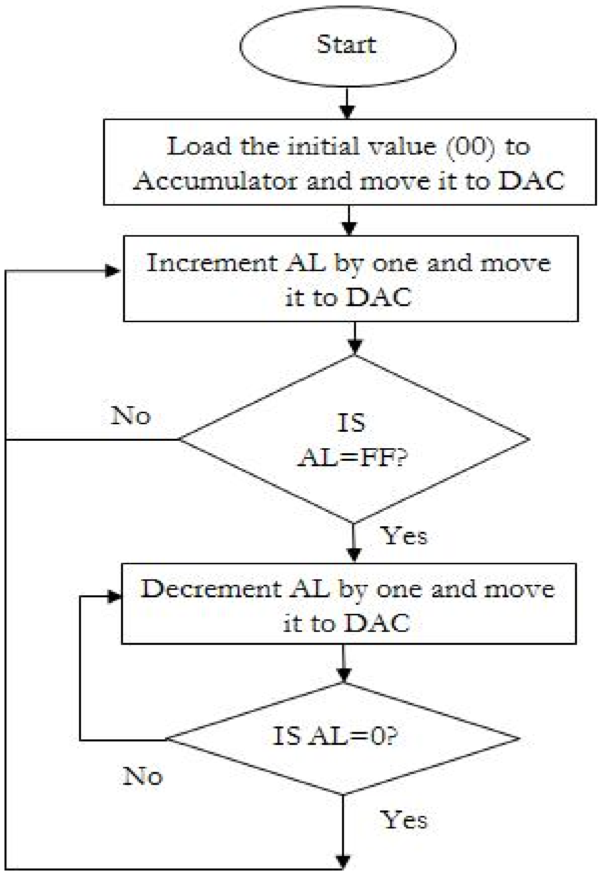
PROGRAM

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

(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


Start
Load the initial value (00) to Accumulator and move it to DAC
Increment AL by one and move it to DAC
No
IS AL=FF?
Yes
Decrement AL by one and move it to DAC
IS AL=0?
No
Yes

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		

<u>REVIEW QUSETIONS:</u>

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 waveform generation using DAC were executed.

Ex. No. : 9
Date:

INTERFACING KEYBOARD AND LCD MATRIX KEYBOARD PROGRAM

AIM:

To write an 8086 assembly language program to interface the 8279 and display the register number, as rolling message.

ALGORITHM:

1. Set the pointer to 1200H
2. Initialize the counter (CX – Reg) to 0FH
3. Send Mode Display Command word (10H) to C2H.
4. Send Clear Display Command word (0CCH) to C2H.
5. Send Write Display Command Word (90H) to C2H
6. Get the data pointed by pointer
7. Output it to C0H
8. Call a Delay program for Lively display
9. Increment memory pointer to point next data.
10. Decrement count.
11. Check if Count is zero. If yes go to step 1. Else go to step 6

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.

Program

Label	Program	Comments
START:	ORG 1000H	Set starting address as 1000H.
	MOV SI, 1200H	Set Pointer
	MOV CX, 000FH	Initialize counter.
	MOV AL, 10H	Set Mode and Display
	OUT C2H, AL	
	MOV AL, 0CCH	Clear display.
	OUT C2H, AL	
	MOV AL, 90H	Write Display
	OUT C2H, AL	
NXTCHR:	MOV AL, [SI]	Get the data
	OUT C0H, AL	
	CALL DELAY	Call Delay program for lively display
	INC SI	Increment Pointer
	LOOP NXTCHR	Decrement Count, If not zero go to NXTCHR
	JMP START	
DELAY:	MOV DX, 0A0FFH	
LOOP1:	DEC DX	
	JNZ LOOP1	
	RET	

DISPLAY MODE SETUP:

0	**0**	**0**	**D**	**D**	**K**	**K**	**K**

DD		**DISPLAY MODE**
00	8	8-bit character display – Left Entry
01	16	8-bit character display – Left Entry
10	8	8-bit character display – Right Entry
11	6	8-bit character display – Right Entry

KKK	**KEYBOARD MODE**
000	Encoded Scan Keyboard – 2 key lock out
001	Decoded Scan Keyboard – 2 key lock out
010	Encoded Scan Keyboard – N Key Roll Over
011	Decoded Scan Keyboard – N Key Roll Over
100	Encoded Scan Sensor Matrix
101	Decoded Scan Sensor Matrix
110	Strobed input, Encoded Display Scan
111	Strobed input, Decoded Display Scan

CLEAR DISPLAY:

1	**1**	**0**	**CD**	**CD**	**CD**	**CF**	**CA**

CD CD CD **-** The lower two CD bits specify the blanking code to be sent to the segments to turn them OFF while the 8279 is switching from one digit to next

CD CD CD

Enables clear display when CD = 1.
The rows of display RAM are cleared by the code set by lower two CD Bits.
If CD = 0 then the contents of RAM will be displayed

0 X	A0 – 3	B0 – 3 = 00 = 0000 0000
1 0	A0 – 3	B0 – 3 = 20 = 0010 0000
1 1	A0 – 3	B0 – 3 = FF = 1111 1111

CF – If CF = 1, FIFO status is cleared, Interrupt output line is reset. Sensor RAM pointer is set to row 0.

CA – Clear All bit has the combined effect of CD and CF. It uses CD clearing code on Display RAM and clears FIFO status.

WRITE DISPLAY RAM:

1	**1**	**0**	**AI**	**A**	**A**	**A**	**A**

AI – Auto Increment Flag. If AI = 1, the row address selected will be incremented after each read or write to the Display RAM.
AAA – Selects one of the 16 rows of the display RAM.

Example:

Input Data:

```
1200:FF
1201:FF
1202:28
1203:0C
1204:1A
1295:FF
1206:98
1207:68
1208:7C
1209:C8
120A:FF
120B:1C
120C:29
120D:F7
120E:FF
```

Output:

GOD HELP US

Observation:

Input Data:

```
1200:
1201:
1202:
1203:
1204:
1295:
1206:
1207:
1208:
1209:
120A:
120B:
120C:
120D:
120E:
120F:
```

Output:

REVIEW QUSETIONS:

1. What is the size of flag register?
2. Can you perform 32 bit operation with 8086? How?
3. What is the difference between instructions DIV & IDIV?
4. What is the size of each segment?
5. What is the difference between instructions MUL & IMUL?
6. What is meant by LED/LCD?
7. How do you place a specific value in DPTR register? (Dec 2013)
8. Which of the 8051 ports need pull-up registers to functions as I/O port ? (Dec 2013)
9. What are the control words of 8251A and what are its functions?
10. What are the display modes supported by the 8279 chip?
11. Give the format of program clock word of 8279 and mention its purpose.
12. What is 2 key lockout and n key rollover?
13. Define – PPI
14. What is the use of direction flag?
15. What are the alternate functions of port0, port1, port2 and port3?

Result:

Thus the program to display the register number, as rolling message, in the display by interfacing 8279 with 8086 was done successfully.

Ex. No. 10
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

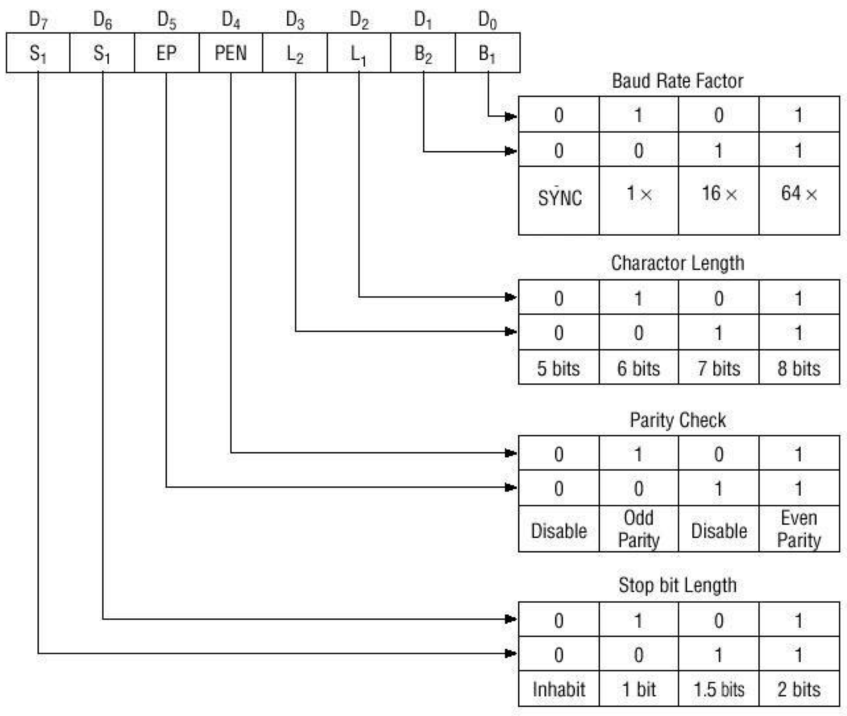
SERIAL INTERFACE

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.

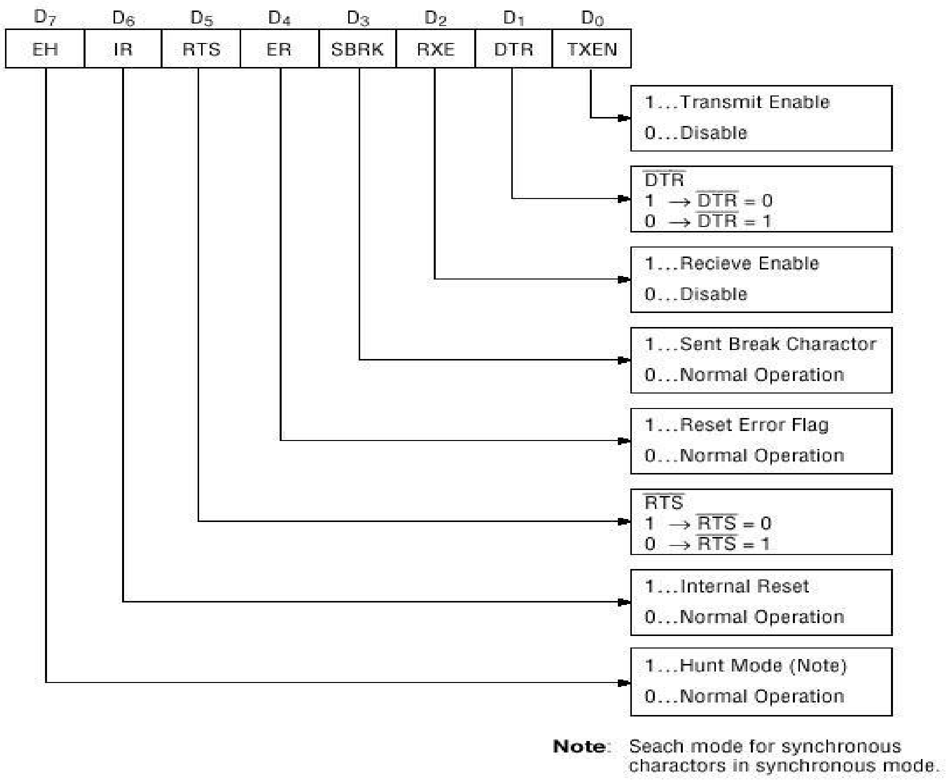
Algorithm:

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.



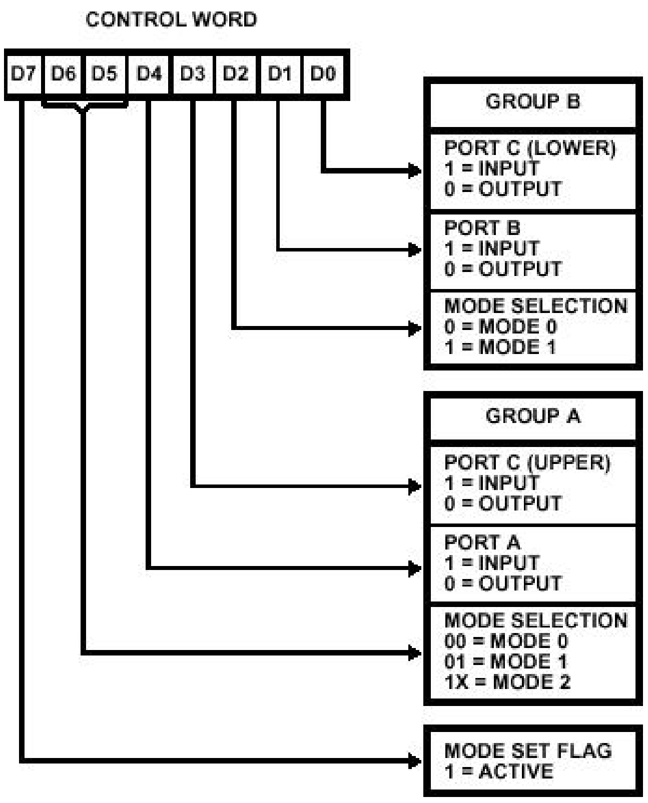
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
S_1 S_1 EP PEN L_2 L_1 B_2 B_1
Baud Rate Factor
0 1 0 1
0 0 1 1
SYNC 1× 16× 64×
Charactor Length
0 1 0 1
0 0 1 1
5 bits 6 bits 7 bits 8 bits
Parity Check
0 1 0 1
0 0 1 1
Disable Odd Parity Disable Even Parity
Stop bit Length
0 1 0 1
0 0 1 1
Inhabit 1 bit 1.5 bits 2 bits

Bit Configuration of Mode Instruction (Asynchronous)



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:

<u>REVIEW QUSETIONS:</u>

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
	OUT C6,AL	input port and port B as output port.
	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 QUSETIONS:

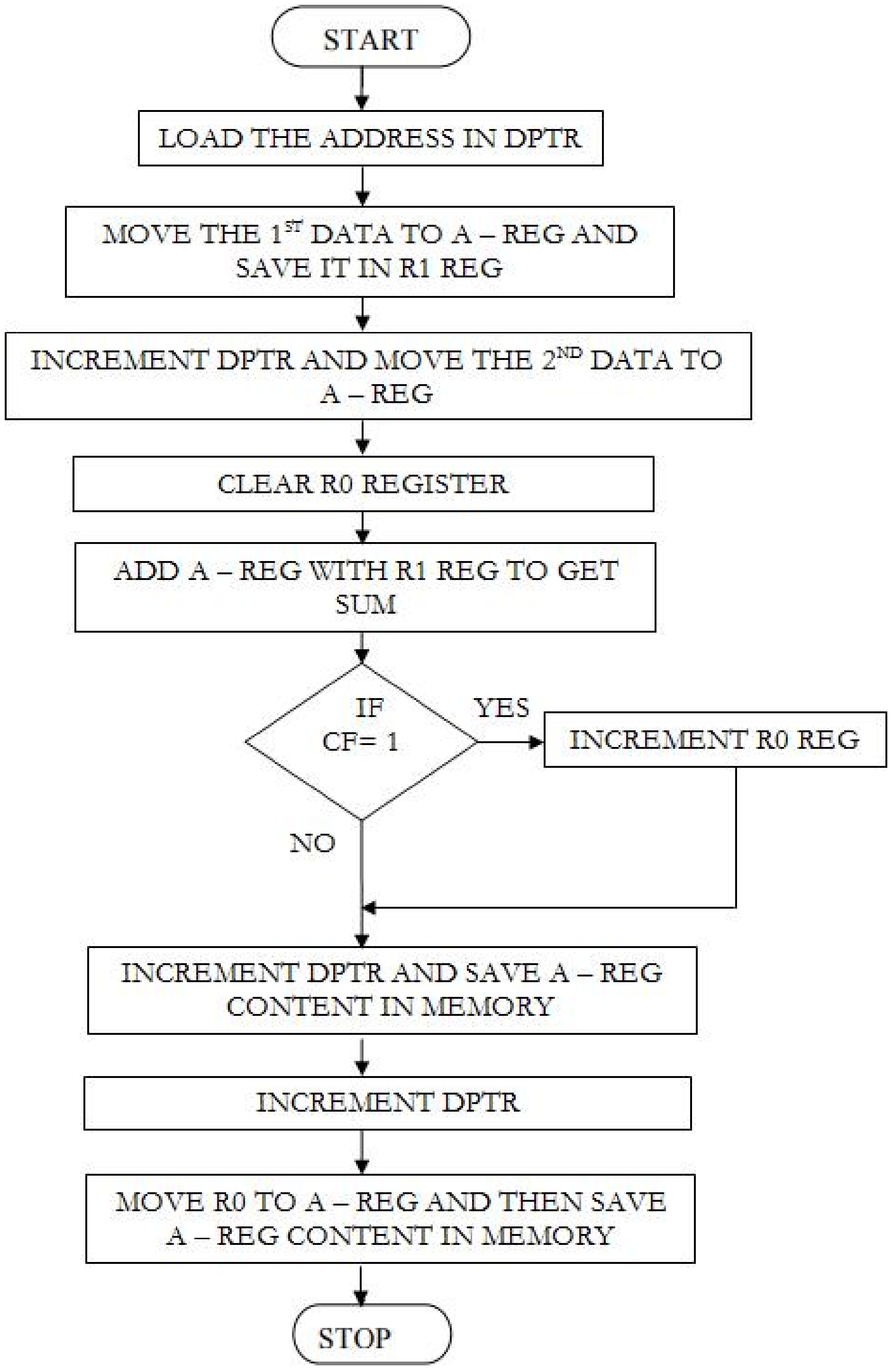
1. What is the difference between near and far procedure?
2. What is difference between shifts and rotate instructions?
3. Which are strings related instructions?
4. Which are addressing modes and their examples in 8086?
5. Discuss the use of following instructions:
   a. SCASB
   b. LAHF
   c. ROL
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

Result:

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

CYCLE II
8051 Programs

Flow Chart


START
LOAD THE ADDRESS IN DPTR
MOVE THE 1ST DATA TO A – REG AND SAVE IT IN R1 REG
INCREMENT DPTR AND MOVE THE 2ND DATA TO A – REG
CLEAR R0 REGISTER
ADD A – REG WITH R1 REG TO GET SUM
IF CF= 1
YES
INCREMENT R0 REG
NO
INCREMENT DPTR AND SAVE A – REG CONTENT IN MEMORY
INCREMENT DPTR
MOVE R0 TO A – REG AND THEN SAVE A – REG CONTENT IN MEMORY
STOP

Ex. No. 11
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 1st data in DPTR
	MOVX A,@DPTR	Move the 1st data to A
	MOV R1,A	Save the first data in R1
	INC DPTR	Increment DPTR to point 2nd data
	MOVX A,@DPTR	Load the 2nd 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

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

Example: **Manual Calculation:**

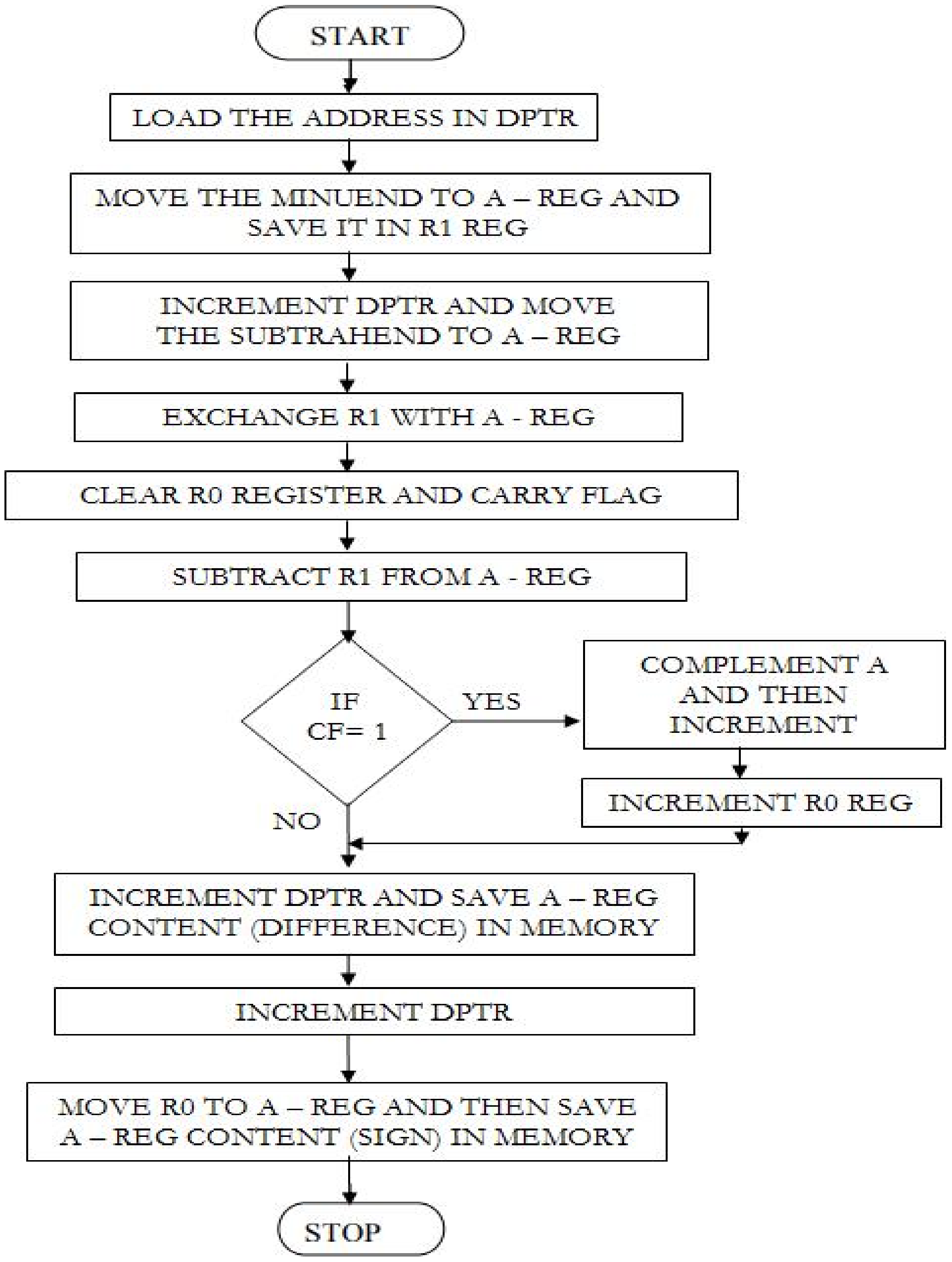
Input:

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

Output:

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

Flow Chart


START
LOAD THE ADDRESS IN DPTR
MOVE THE MINUEND TO A – REG AND SAVE IT IN R1 REG
INCREMENT DPTR AND MOVE THE SUBTRAHEND TO A – REG
EXCHANGE R1 WITH A - REG
CLEAR R0 REGISTER AND CARRY FLAG
SUBTRACT R1 FROM A - REG
IF CF= 1
YES
COMPLEMENT A AND THEN INCREMENT
INCREMENT R0 REG
NO
INCREMENT DPTR AND SAVE A – REG CONTENT (DIFFERENCE) IN MEMORY
INCREMENT DPTR
MOVE R0 TO A – REG AND THEN SAVE A – REG CONTENT (SIGN) IN MEMORY
STOP

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: **Manual Calculation:**

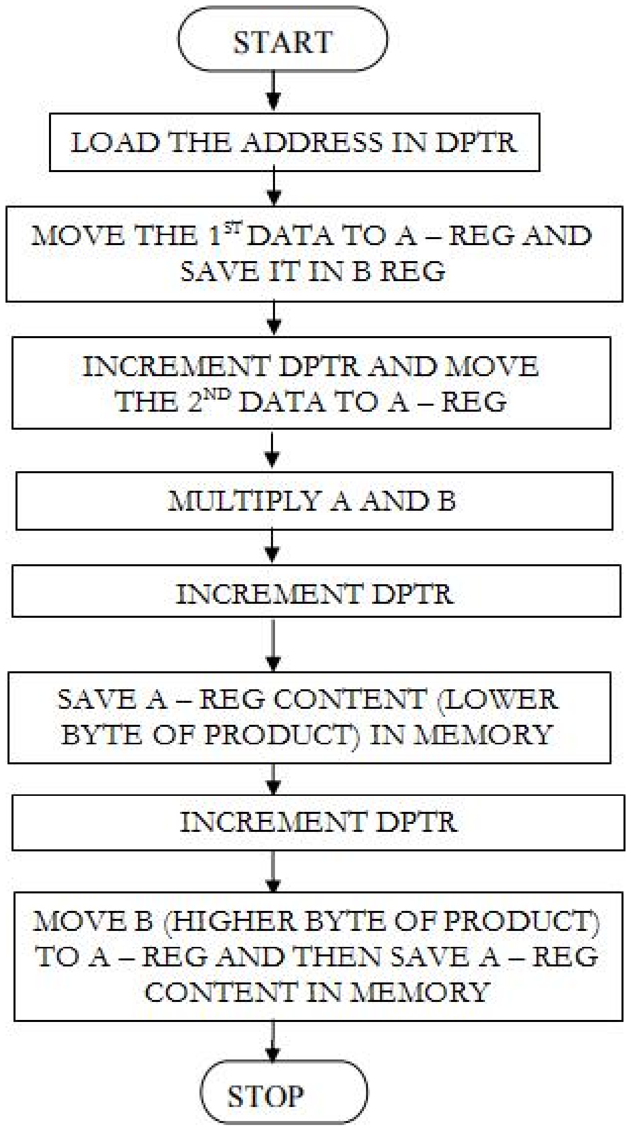
Input:

4500: 0A	[Minuend]
4501:05	[Subtrahend]

Output:

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

Flow Chart



START
LOAD THE ADDRESS IN DPTR
MOVE THE 1ST DATA TO A – REG AND SAVE IT IN B REG
INCREMENT DPTR AND MOVE THE 2ND DATA TO A – REG
MULTIPLY A AND B
INCREMENT DPTR
SAVE A – REG CONTENT (LOWER BYTE OF PRODUCT) IN MEMORY
INCREMENT DPTR
MOVE B (HIGHER BYTE OF PRODUCT) TO A – REG AND THEN SAVE A – REG CONTENT IN MEMORY
STOP

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 1st data in DPTR
	MOVX A,@DPTR	Move the 1st data to A
	MOV B,A	Save the 1st data in B
	INC DPTR	Increment DPTR to point 2nd data
	MOVX A,@DPTR	Load the 2nd 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: **Manual Calculation:**

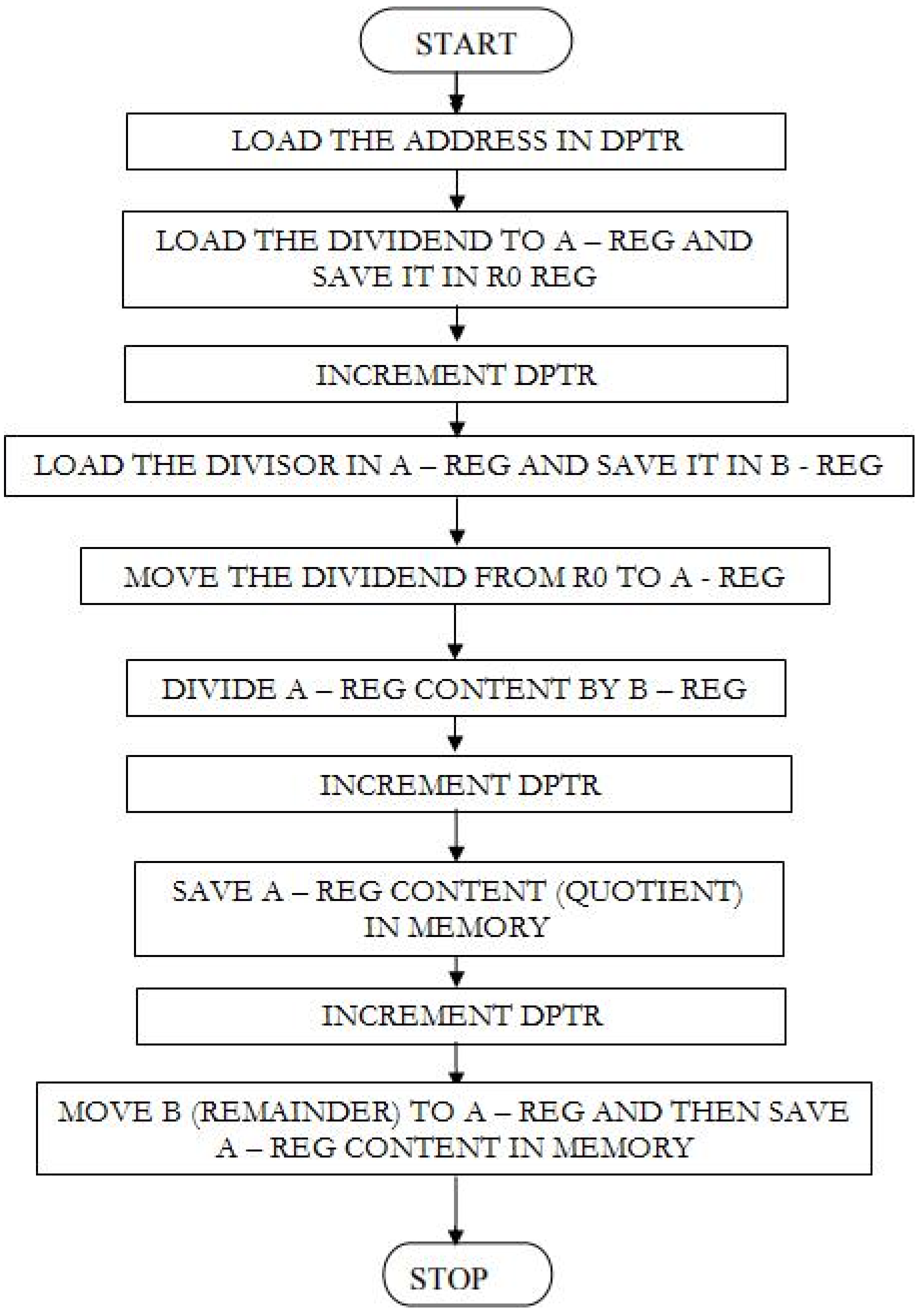
Input:

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

Output:

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

FLOWCHART


START
LOAD THE ADDRESS IN DPTR
LOAD THE DIVIDEND TO A – REG AND SAVE IT IN R0 REG
INCREMENT DPTR
LOAD THE DIVISOR IN A – REG AND SAVE IT IN B - REG
MOVE THE DIVIDEND FROM R0 TO A - REG
DIVIDE A – REG CONTENT BY B – REG
INCREMENT DPTR
SAVE A – REG CONTENT (QUOTIENT) IN MEMORY
INCREMENT DPTR
MOVE B (REMAINDER) TO A – REG AND THEN SAVE A – REG CONTENT IN MEMORY
STOP

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
	MOVX @DPTR,A	Save it in external memory
HERE:	SJMP HERE	Remain idle in infinite loop

Example: **Manual Calculation:**

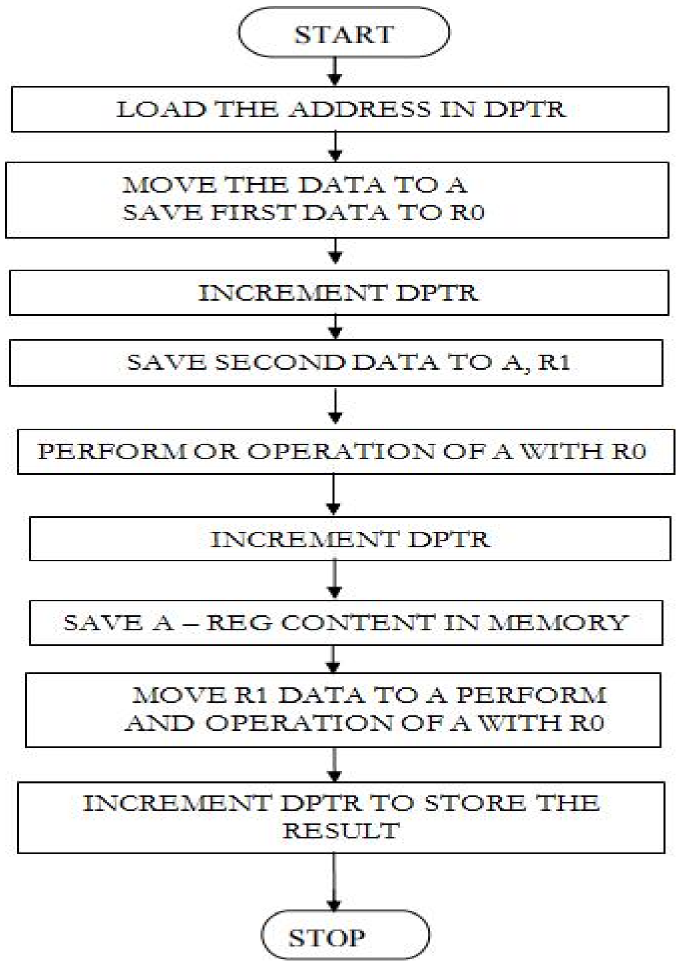
Input:

4500: 04	[Dividend]
4501:02	[Divisor]

Output:

4502:02	[Quotient]
4503:00	[Remainder]

FLOWCHART


START
LOAD THE ADDRESS IN DPTR
MOVE THE DATA TO A
SAVE FIRST DATA TO R0
INCREMENT DPTR
SAVE SECOND DATA TO A, R1
PERFORM OR OPERATION OF A WITH R0
INCREMENT DPTR
SAVE A – REG CONTENT IN MEMORY
MOVE R1 DATA TO A PERFORM
AND OPERATION OF A WITH R0
INCREMENT DPTR TO STORE THE
RESULT
STOP

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:

Manual Calculation:

Input

4500 :00

4501:01

Output

4502 :01 (OR operation)

4503 :00 (AND operation)

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

Example:

Input:

4500:03

Output:

4501:09

4502:00

REVIEW QUSETIONS:

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.

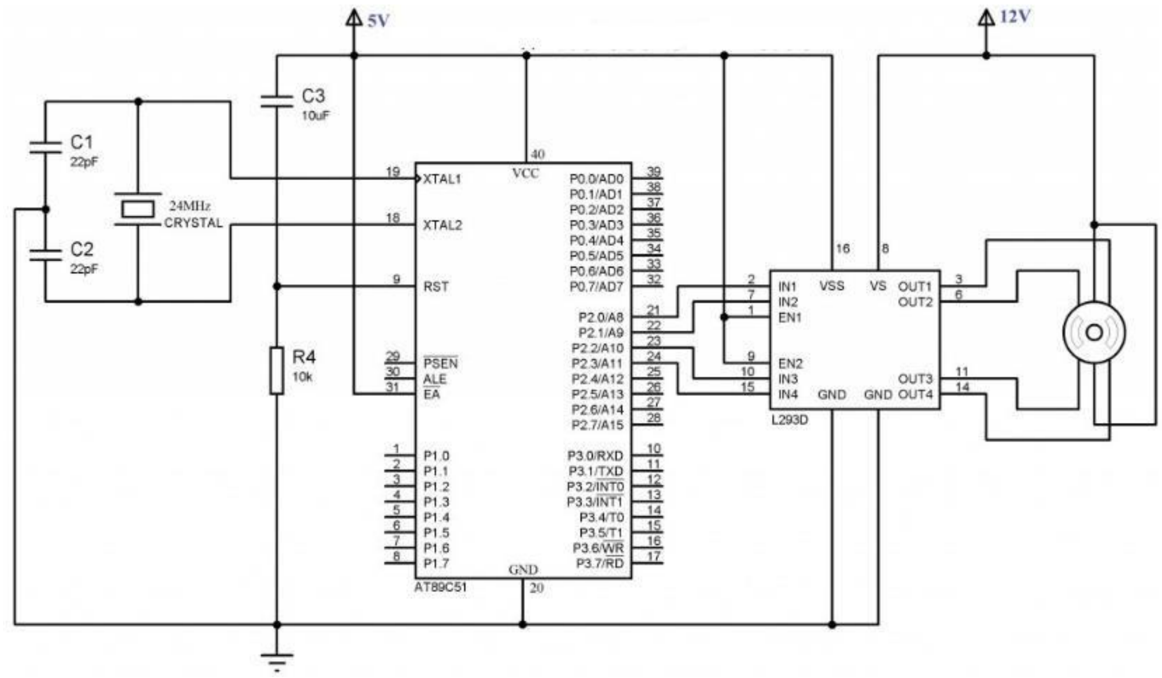
EXPT NO: 12
DATE:

INTERFACING STEEPER MOTOR USING 8051 MICROCONTROLLERS

AIM:
To interface the steeper motor with 8051 microcontrollers.

APPARATUS REQUIRED:
8051 microcontroller kit
L293D motor

Schematic:



Program:

```
#include<reg52.h>

#include<stdio.h>


void delay(int);


void main()

{

  do

  {
    P2 = 0x03; //0011

    delay(1000);

    P2 = 0x06; //0110

    delay(1000);

    P2 = 0x0C; //1100

    delay(1000);

    P2 = 0x09; //1001

    delay(1000);

  }

while(1);

}


void delay(int k)

{

  int i,j;

  for(i=0;i<k;i++)

  {

    for(j=0;j<100;j++)

    {}

  }

}
```

Keil C Code for Half Drive

```
#include<reg52.h>

#include<stdio.h>


void delay(int);
void main()
{
  do
  {
    P2=0x01; //0001
    delay(1000);
    P2=0x03; //0011
    delay(1000);
    P2=0x02; //0010
    delay(1000);
    P2=0x06; //0110
    delay(1000);
    P2=0x04; //0100
    delay(1000);
    P2=0x0C; //1100
    delay(1000);
    P2=0x08; //1000
    delay(1000);
    P2=0x09; //1001
    delay(1000);
  } while(1);
}

void delay(int k)
{
  int i,j;
  for(i=0;i<k;i++)
  {
    for(j=0;j<100;j++)
    {}
  }
}
```

RESULT:

Thus the steeper motor was controlled using 8051 microcontrollers.

EXPT NO: 13
DATE:

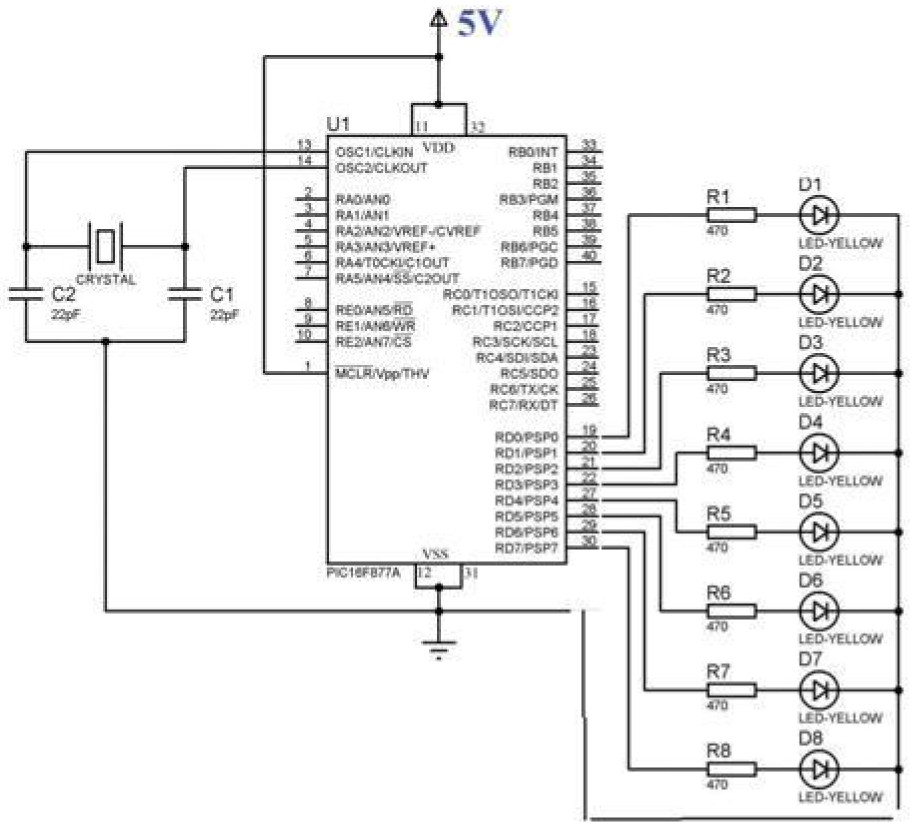
LED BLINKING USING PIC MICROCONTROLLERS

AIM:
To make an LED blinking display using PIC microcontrollers.

APPARATUS REQUIRED:
PIC microcontroller kit

LED BLINKING

Schematic:



Pin Details:

PORT RD(0 – 7) – LED (inbuilt in KIT)

Program:

```
#include <htc.h>

#define _XTAL_FREQ 20000000

void main()

{

  TRISD=0X00;

  PORTD=0X00;

  while(1)

  {

PORTD=0XFF;   // LED on

_delay_ms(1000);  // 1 sec delay

PORTD=0X00;           // LED off

__delay_ms(1000);

  }

}
```

RESULT:

Thus LED blinking was done using PIC microcontrollers.

EXPT NO: 14
DATE:

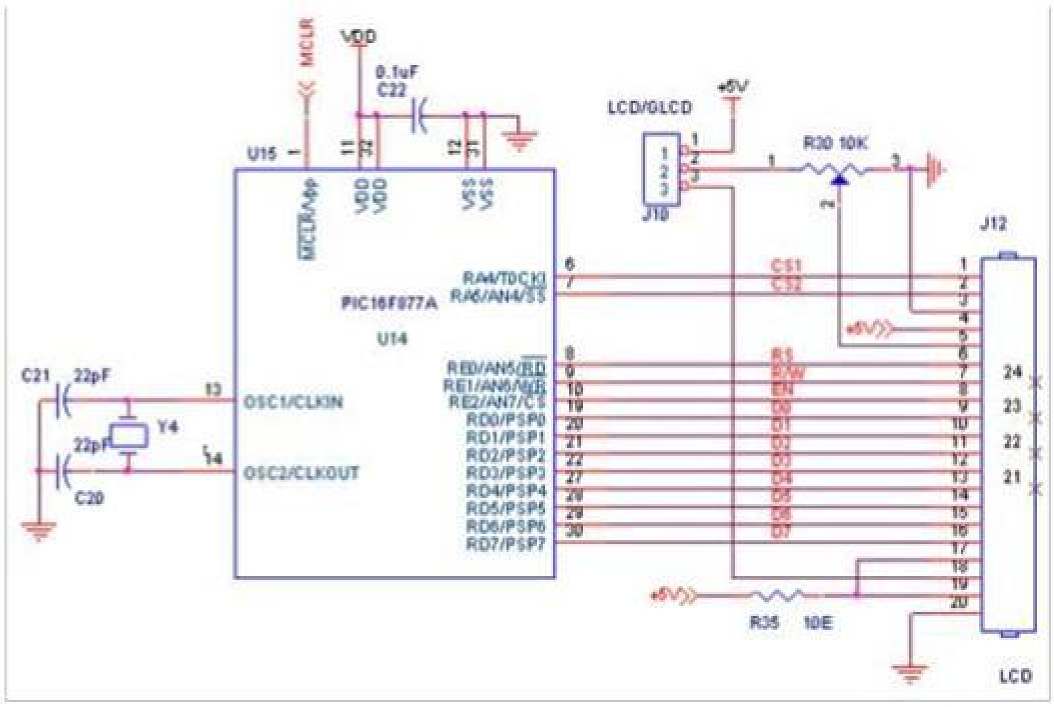
LCD DISPLAY USING PIC MICROCONTROLLERS

AIM:
To interface an LCD display using PIC microcontrollers.

APPARATUS REQUIRED:
PIC microcontroller kit

LCD DISPLAY

Schematic:



Pin Details:

PORT RE0 = RS

PORT RE1 = R/W

PORT RE2 = E

PORT RD(0-7) = DATA(D1-D7).

Program:

```
#include <htc.h>

#define _XTAL_FREQ 20000000
#define rs RE0


#define rw RE1

#define e  RE2

#define data PORTD

#define datadr TRISD

#define contdr TRISE

void adc();

void lcd_int();

void cmd(unsigned char a);

void dat(unsigned char b);

int i;

void main()

{

        ADCON1 = 0x06;

        datadr =0x00;

        contdr =0x00;                    //Port B and Port C is Output (LCD)

        TRISA0=1;                                //RA0 is input (ADC)
```

```
            lcd_int();
            dat('H');
            dat('E');
            dat('L');
            dat('L');
            dat('o');
while(1);


}
void lcd_int()
{
             cmd(0x38);  // for using 2 lines and 5X7 matrix of LCD
    __delay_ms(5);
cmd(0x0e);  // turn display ON, cursor blinking
   __delay_ms(5);
cmd(0x01);  //clear screen
__delay_ms(5);
cmd(0x06);  // bring cursor to position 1 of line 1
   __delay_ms(5);
            cmd(0x80);  // bring cursor to position 1 of line 1


}
void cmd(unsigned char a)
{
            data =a;
            rs=0;
            rw=0;
```

```
        e=1;

        __delay_ms(5);

        e=0;

}

void dat(unsigned char b)

{

        data = b;

        rs=1;

        rw=0;

        e=1;

        __delay_ms(5);

        e=0;
}
```

RESULT:

Thus the LCD interface was done using PIC microcontrollers.

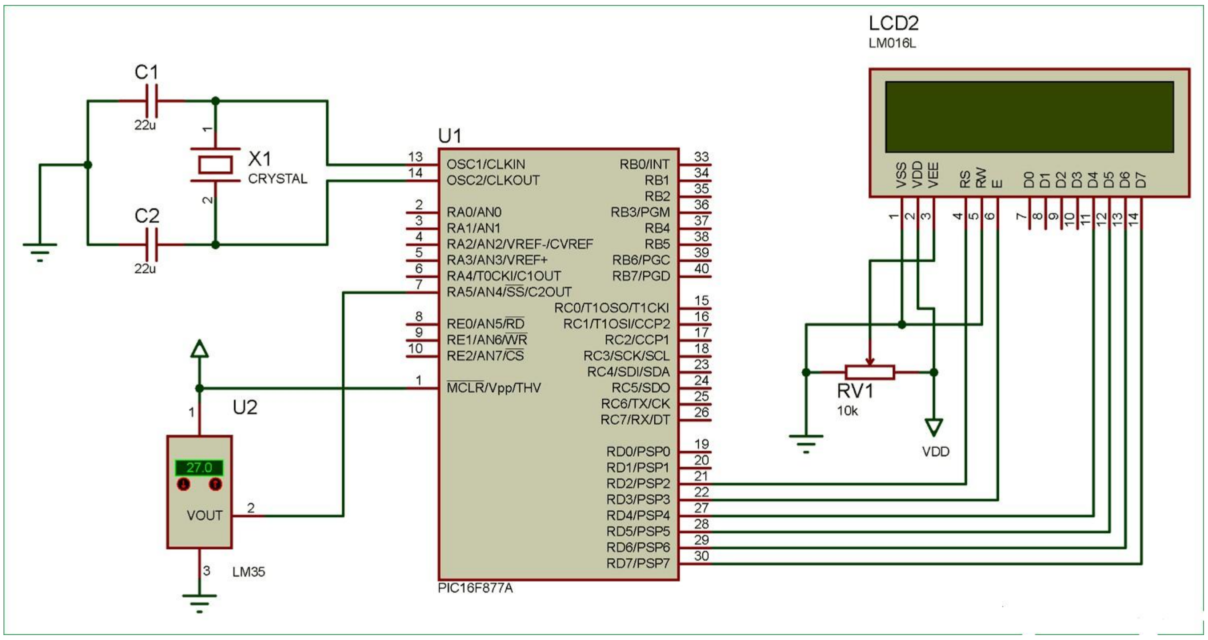
EXPT NO: 15
DATE:

INTERFACING TEMPERATURE SENSOR USING PIC MICROCONTROLLERS

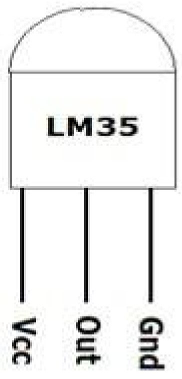
AIM:
To interface the temperature sensor using PIC microcontrollers.

APPARATUS REQUIRED:
PIC microcontroller kit
LM35 Temperature Sensor

Schematic:



Pin Details:



Program:

```
#define _XTAL_FREQ 20000000
#define RS RD2
#define EN RD3
#define D4 RD4
#define D5 RD5
#define D6 RD6
#define D7 RD7
#include <xc.h>
#pragma config FOSC = HS        // Oscillator Selection bits (HS oscillator)

#pragma config WDTE = OFF       // Watchdog Timer Enable bit (WDT disabled)

#pragma config PWRTE = ON       // Power-up Timer Enable bit (PWRT enabled)

#pragma config BOREN = ON       // Brown-out Reset Enable bit (BOR enabled)

#pragma config LVP = OFF        // Low-Voltage (Single-Supply) In-Circuit Serial
Programming Enable bit (RB3 is digital I/O, HV on MCLR must be used for programming)

#pragma config CPD = OFF        // Data EEPROM Memory Code Protection bit (Data
EEPROM code protection off)

#pragma config WRT = OFF        // Flash Program Memory Write Enable bits (Write
protection off; all program memory may be written to by EECON control)

#pragma config CP = OFF         // Flash Program Memory Code Protection bit (Code
protection off)


void ADC_Initialize()

{
  ADCON0 = 0b01000001; //ADC ON and Fosc/16 is selected

  ADCON1 = 0b11000000; // Internal reference voltage is selected

}


unsigned int ADC_Read(unsigned char channel)

{

  ADCON0 &= 0x11000101; //Clearing the Channel Selection Bits

  ADCON0 |= channel<<3; //Setting the required Bits

  __delay_ms(2); //Acquisition time to charge hold capacitor

  GO_nDONE = 1; //Initializes A/D Conversion

  while(GO_nDONE); //Wait for A/D Conversion to complete

  return ((ADRESH<<8)+ADRESL); //Returns Result

}



//LCD Functions Developed by Circuit Digest.

void Lcd_SetBit(char data_bit) //Based on the Hex value Set the Bits of the Data Lines

{

    if(data_bit& 1)

        D4 = 1;

    else

        D4 = 0;


    if(data_bit& 2)

        D5 = 1;

    else
```

```
        D5 = 0;

    if(data_bit& 4)
        D6 = 1;
    else
        D6 = 0;

    if(data_bit& 8)
        D7 = 1;
    else
        D7 = 0;
}

void Lcd_Cmd(char a)
{
    RS = 0;
    Lcd_SetBit(a); //Incoming Hex value
    EN  = 1;
        _delay_ms(4);
        EN  = 0;
}

Lcd_Clear()
{
    Lcd_Cmd(0); //Clear the LCD
    Lcd_Cmd(1); //Move the curser to first position
```

```
}

void Lcd_Set_Cursor(char a, char b)

{

  char temp,z,y;

  if(a== 1)

  {

   temp = 0x80 + b - 1; //80H is used to move the curser

    z = temp>>4; //Lower 8-bits

    y = temp & 0x0F; //Upper 8-bits

    Lcd_Cmd(z); //Set Row

    Lcd_Cmd(y); //Set Column

  }

  else if(a== 2)

  {

    temp = 0xC0 + b - 1;

    z = temp>>4; //Lower 8-bits

    y = temp & 0x0F; //Upper 8-bits

    Lcd_Cmd(z); //Set Row

    Lcd_Cmd(y); //Set Column

  }

}

void Lcd_Start()

{
  Lcd_SetBit(0x00);

  for(int i=1065244; i<=0; i--)  NOP();

  Lcd_Cmd(0x03);

    _delay_ms(5);

  Lcd_Cmd(0x03);

    _delay_ms(11);

  Lcd_Cmd(0x03);

  Lcd_Cmd(0x02); //02H is used for Return home -> Clears the RAM and initializes the LCD

  Lcd_Cmd(0x02); //02H is used for Return home -> Clears the RAM and initializes the LCD

  Lcd_Cmd(0x08); //Select Row 1

  Lcd_Cmd(0x00); //Clear Row 1 Display

  Lcd_Cmd(0x0C); //Select Row 2

  Lcd_Cmd(0x00); //Clear Row 2 Display

  Lcd_Cmd(0x06);

}


void Lcd_Print_Char(char data)  //Send 8-bits through 4-bit mode

{

   char Lower_Nibble,Upper_Nibble;

   Lower_Nibble = data&0x0F;

   Upper_Nibble = data&0xF0;

   RS = 1;             // => RS = 1

   Lcd_SetBit(Upper_Nibble>>4);             //Send upper half by shifting by 4

   EN = 1;

   for(int i=2130483; i<=0; i--)  NOP();
   EN = 0;

   Lcd_SetBit(Lower_Nibble); //Send Lower half

   EN = 1;

   for(int i=2130483; i<=0; i--)  NOP();

   EN = 0;

}

void Lcd_Print_String(char *a)

{

    int i;

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

       Lcd_Print_Char(a[i]);  //Split the string using pointers and call the Char function

}


int main()

{

    float adc;

    float volt, temp;

    int c1, c2, c3, c4, temp1;

    ADC_Initialize();


    unsigned int a;

    TRISD = 0x00;

    Lcd_Start();

    while(1)
    {

        adc = (ADC_Read(4)); // Reading ADC values

        volt = adc*4.88281; // Convert it into the voltage

        temp=volt/10.0;  // Getting the temperature values

        temp1 = temp*100;



        c1 = (temp1/1000)%10;

        c2 = (temp1/100)%10;

        c3 = (temp1/10)%10;

        c4 = (temp1/1)%10;
Lcd_Clear();
   Lcd_Set_Cursor(1,3);
        Lcd_Print_String("Temperature");
        Lcd_Set_Cursor(2,5);
        Lcd_Print_Char(c1+'0');
        Lcd_Print_Char(c2+'0');
        Lcd_Print_String(".");
        Lcd_Print_Char(c3+'0');

        Lcd_Print_Char(c4+'0');

        Lcd_Print_Char(0xDF);

        Lcd_Print_String("C");

        __delay_ms(3000);

    }

    return 0;
}
```

RESULT:

Thus the temperature sensor was interfaced using PIC microcontrollers.

EXPT NO: 16
DATE:

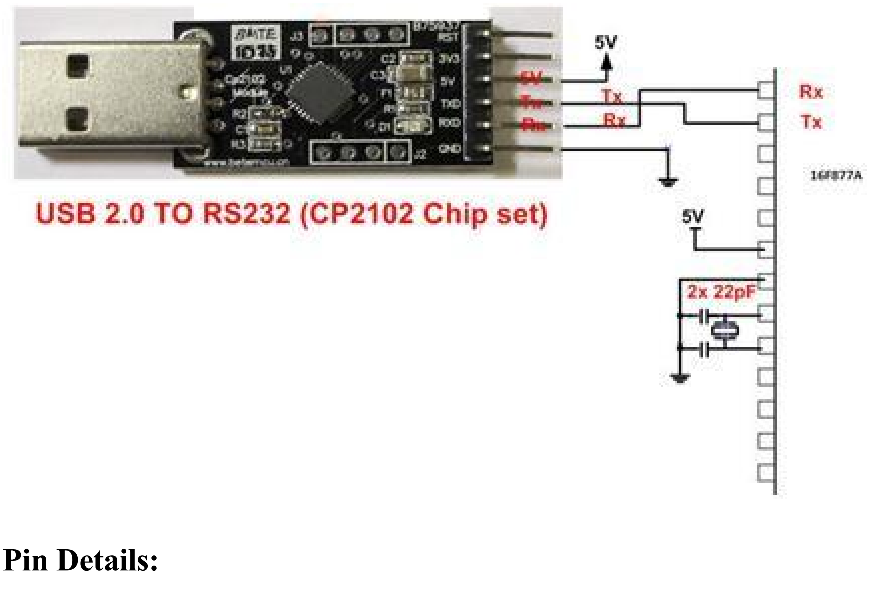
SERIAL COMMUNICATION USING PIC MICROCONTROLLERS

AIM:
To establish serial communication using PIC microcontrollers.

APPARATUS REQUIRED:
PIC microcontroller with RS232 interface

SERIAL COMMUNICATION

Schematic:



Pin Details:

Connect the USB convert to PC and use X-CTU to check the output

Program:

```
#define _XTAL_FREQ 20000000
```

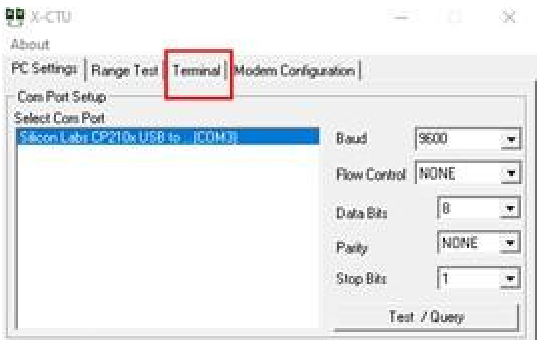
```
#include <htc.h>

void UART_Write(char data);

void UART_Write_Text(char *text);

void main()

{

  TRISC = 0x80; //RC7(RX) as input and RC6 (TX) as output

  SPBRG = 129; // 9600 baud rate

  RCSTA = 0x80;//serial port enable

  BRGH = 1;// high speed baud rate

  TXEN = 1;//enable transmission

__delay_ms(100);

UART_Write_Text("PIC 16F877A SERIAL PROGRAM");

            while(1);

}

void UART_Write(char data)

{

  while(!TRMT);// check TXREG empty

  TXREG = data;

  __delay_ms(5);

}

void UART_Write_Text(char *text)

{

int i;

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

              UART_Write(text[i]);

}

char UART_Read()
```

```
{


  while(!RCIF);

  return RCREG;

}

void UART_Read_Text(char *Output, unsigned int length)

{

            unsigned int i;

            for(int i=0;i<length;i++)

                        Output[i] = UART_Read();

}
```

Note: To check the output

Connect the serial converter between the trainer kit and PC.

Open the X-CTU software and select the com in which the kit is connected and let the parameters as given below.



Com Port Setup
Select Com Port
Silicon Labs CP210x USB to ... (COM3)

Baud 9600
Flow Control NONE
Data Bits 8
Parity NONE
Stop Bits 1

Finally select the terminal pallet on the top marked in the read box then reset the kit once. Check the output text which will be displayed in the terminal. Press reset to print the text again in the terminal.

RESULT:

Thus serial communication was established using PIC microcontrollers.