MICROPROCESSOR LAB MANUAL

Department of Electronics & Communication Engineering
ABOUT THE MANUAL & PREPARATION OF RECORD

- This Manual contains the program of 8085, 8086, 8051 and its interfacing enlisted in the syllabus of CS 2259 microprocessor and microcontroller lab.

- Aim of the experiment is also given at the beginning of each experiment. Once the student is able to write the program as per the algorithm. He/she is supposed to go through the instruction sets carefully and do the experiment step by step.

- It is also expected that the students prepare the theory relevant to the experiment referring to prescribed reference books/journals, in advance, and carry out the experiment after understanding thoroughly the concept and algorithm of the experiment.

- They should get their observations verified and signed by the staff as soon as the experiments are completed and prepare & submit the record of the experiment while they come for the laboratory in the subsequent week.

- The record should contain Experiment No., Date, Aim, Algorithm, program, flowchart, theory, observation and Result.

- The students are directed to discuss & clarify their doubts with the staff members as and when required. They are also directed to follow strictly the guidelines specified.
SYLLABUS

LIST OF EXPERIMENTS

1. Programming with 8085.

2. Programming with 8086-experiments including BIOS/DOS calls, keyboard control, Display, File Manipulation.

3. Interfacing with 8085/8086-8255, 8253.

4. Interfacing with 8085/8086-8279, 8251.

5. 8051 Microcontroller based experiments for control Applications.

6. Mini Project.
## CS 2259 MICROPROCESSOR LABORATORY

### PROGRESS OF THE LABORATORY CLASSES

<table>
<thead>
<tr>
<th>S.No</th>
<th>Name of Experiment</th>
<th>Date of Experiment</th>
<th>Date of Submission</th>
<th>Marks awarded out of 100</th>
<th>Signature of the staff</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Addition of two 8-bit numbers</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>Subtraction of two 8-bit numbers</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>Multiplication of two 8-bit numbers</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>Division of two 8-bit numbers</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>Addition of two 16-bit numbers</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td>Subtraction of two 16-bit numbers</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.</td>
<td>Multiplication of two 16-bit numbers</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.</td>
<td>Division of two 16-bit numbers</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9.</td>
<td>Largest Elements in an array of Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.</td>
<td>Smallest Elements in an array of Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11.</td>
<td>Arrange an array of data in Ascending order</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12.</td>
<td>Arrange an array of data in Descending order</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13.</td>
<td>Code Conversion – Decimal to Hexadecimal</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14.</td>
<td>Code Conversion – Hexadecimal to Decimal</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15.</td>
<td>BCD Addition</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16.</td>
<td>BCD Subtraction</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Programming with 8085
# Programming with 8086

1. BIOS / DOS Calls – Display
2. BIOS / DOS Calls – File Manipulation
3. BIOS / DOS Calls – Disk Information
4. String Manipulation – Search a Word
5. String Manipulation – Find and Replace a Word
6. String Manipulation – Copy a String
7. String Manipulation – Sorting

# Interfacing with 8085

1. Interfacing 8255 (PPI) with 8085
2. Interfacing 8253 Timer with 8085
3. Interfacing 8279 Keyboard/Display Controller with 8085
4. Interfacing 8251 (USART) with 8085

# Programming with 8051

1. Sum of Elements in an array
2. Hexadecimal to Decimal conversion
3. Decimal to Hexadecimal Conversion
4. Stepper Motor Interfacing with 8051
INTRODUCTION TO 8085

8085 Architecture Pin Diagram

--- X1 [1] 40 | Vcc (+5V)
--- X2 [2] 39 | HOLD <=
<-- RESET OUT [3] 38 | HLDA -->
<-- SOD [4] 37 | CLK (OUT) -->
--- SID [5] 36 | RESET IN <=
--- TRAP [6] 35 | READY <=
--- RST 7.5 [7] 34 | IO/M -->
--- RST 6.5 [8] 33 | S1 -->
--- RST 5.5 [9] 32 | RD -->
--- INTR [10] 8085A 31 | WR -->
<-- AD0 [12] 29 | S0 -->
<-- AD1 [13] 28 | A15 -->
<-- AD2 [14] 27 | A14 -->
<-- AD3 [15] 26 | A13 -->
<-- AD4 [16] 25 | A12 -->
<-- AD5 [17] 24 | A11 -->
<-- AD6 [18] 23 | A10 -->
<-- AD7 [19] 22 | A9 -->
(Gnd) Vss [20]--------------------

21 | A8 -->
ARCHITECTURE or FUNCTIONAL BLOCK DIAGRAM OF 8085
Introduction:

INTEL 8085 is one of the most popular 8-bit microprocessor capable of addressing 64 KB of memory and its architecture is simple. The device has 40 pins, requires +5 V power supply and can operate with 3MHz single phase clock.

ALU (Arithmetic Logic Unit):

The 8085A has a simple 8-bit ALU and it works in coordination with the accumulator, temporary registers, 5 flags and arithmetic and logic circuits. ALU has the capability of performing several mathematical and logical operations. The temporary registers are used to hold the data during an arithmetic and logic operation. The result is stored in the accumulator and the flags are set or reset according to the result of the operation. The flags are affected by the arithmetic and logic operation. They are as follows:

- **Sign flag**: After the execution of the arithmetic - logic operation if the bit D7 of the result is 1, the sign flag is set. This flag is used with signed numbers. If it is 1, it is a negative number and if it is 0, it is a positive number.

- **Zero flag**: The zero flag is set if the ALU operation results in zero. This flag is modified by the result in the accumulator as well as in other registers.

- **Auxiliary carry flag**: In an arithmetic operation when a carry is generated by digit D3 and passed on to D4, the auxiliary flag is set.

- **Parity flag**: After arithmetic – logic operation, if the result has an even number of 1’s the flag is set. If it has odd number of 1’s it is reset.

- **Carry flag**: If an arithmetic operation results in a carry, the carry flag is set. The carry flag also serves as a borrow flag for subtraction.
**Timing and control unit:** This unit synchronizes all the microprocessor operation with a clock and generates the control signals necessary for communication between the microprocessor and peripherals. The control signals RD (read) and WR (write) indicate the availability of data on the data bus.

**Instruction register and decoder:** The instruction register and decoder are part of the ALU. When an instruction is fetched from memory it is loaded in the instruction register. The decoder decodes the instruction and establishes the sequence of events to follow.

**Register array:** The 8085 has six general purpose registers to store 8-bit data during program execution. These registers are identified as B, C, D, E, H and L. They can be combined as BC, DE and HL to perform 16-bit operation.

**Accumulator:** Accumulator is an 8-bit register that is part of the ALU. This register is used to store 8-bit data and to perform arithmetic and logic operation. The result of an operation is stored in the accumulator.

**Program counter:** The program counter is a 16-bit register used to point to the memory address of the next instruction to be executed.

**Stack pointer:** It is a 16-bit register which points to the memory location in R/W memory, called the Stack.

**Communication lines:** 8085 microprocessor performs data transfer operations using three communication lines called buses. They are address bus, data bus and control bus.

- **Address bus** – it is a group of 16-bit lines generally identified as A_0 – A_15. The address bus is unidirectional i.e., the bits flow in one direction from microprocessor to the peripheral devices. It is capable of addressing 2^{16} memory locations.
- Data bus – it is a group of 8 lines used for data flow and it is bidirectional. The data ranges from 00 – FF.
- Control bus – it consist of various single lines that carry synchronizing signals. The microprocessor uses such signals for timing purpose.

Review Questions:

1. What are the various registers in 8085?
2. In 8085 name the 16 bit registers?
3. What are the various flags used in 8085?
4. What is Stack Pointer?
5. What is Program counter?
6. Which Stack is used in 8085?
7. What happens when HLT instruction is executed in processor?
8. What is meant by a bus?
9. What is Tri-state logic?
10. Give an example of one address microprocessor?
11. In what way interrupts are classified in 8085?
12. What are Hardware interrupts?
13. What are Software interrupts?
14. Which interrupt has the highest priority?
15. Name 5 different addressing modes?
16. How many interrupts are there in 8085?
17. What is clock frequency for 8085?
18. What is the RST for the TRAP?
19. In 8085 which is called as High order / Low order Register?
20. What are input & output devices?
21. Can an RC circuit be used as clock source for 8085?
22. Why crystal is a preferred clock source?
23. Which interrupt is not level-sensitive in 8085?.
24. What does Quality factor mean?
25. What are level-triggering interrupt?
Introduction to 8086

PIN DIAGRAM

40 Lead

231455-2
ARCHITECTURAL DIAGRAM OF 8086
Review Questions:

1. What are the flags in 8086?
2. What are the various interrupts in 8086?
3. What is meant by Maskable interrupts?
4. What is Non-Maskable interrupts?
5. Which interrupts are generally used for critical events?
6. Give examples for Maskable interrupts?
7. Give example for Non-Maskable interrupts?
8. What is the Maximum clock frequency in 8086?
9. What are the various segment registers in 8086?
10. Which Stack is used in 8086?
11. What are the address lines for the software interrupts?
12. What is SIM and RIM instructions?
13. Which is the tool used to connect the user and the computer?
14. What is the position of the Stack Pointer after the PUSH instruction?
15. What is the position of the Stack Pointer after the POP instruction?
16. Logic calculations are done in which type of registers?
17. What are the different functional units in 8086?
18. Give examples for Micro controller?
19. What is meant by cross-compiler?
20. What are the address lines for the hardware interrupts?

21. Which Segment is used to store interrupt and subroutine return address registers?

22. Which Flags can be set or reset by the programmer and also used to control the operation of the processor?
23. What does EU do?
24. Which microprocessor accepts the program written for 8086 without any changes?
25. What is the difference between 8086 and 8088?
# Introduction to Microcontroller

## 8051

- A smaller computer
- On-chip RAM, ROM, I/O ports...
- Example: Motorola’s 6811, Intel’s 8051, Zilog’s Z8 and PIC 16X

## Microprocessor vs. Microcontroller

<table>
<thead>
<tr>
<th>Microprocessor</th>
<th>Microcontroller</th>
</tr>
</thead>
<tbody>
<tr>
<td>- CPU is stand-alone, RAM, ROM, I/O, timer are separate</td>
<td>- CPU, RAM, ROM, I/O and timer are all on a single chip</td>
</tr>
<tr>
<td>- designer can decide on the amount of ROM, RAM and I/O ports.</td>
<td>- fix amount of on-chip ROM, RAM, I/O ports</td>
</tr>
<tr>
<td>- expansive</td>
<td>- for applications in which cost, power and space are critical</td>
</tr>
<tr>
<td>- versatility</td>
<td>- single-purpose</td>
</tr>
<tr>
<td>- general-purpose</td>
<td></td>
</tr>
</tbody>
</table>
Block Diagram

External interrupts

Interrupt Control

On-chip ROM for program code

CPU

On-chip RAM

Bus Control

4 I/O Ports

Serial Port

Counter Inputs

Timer/Counter

Timer 1

Timer 0

Pin Description of the 8051

8051

(8031)

Vcc

P0.0(AD0)

P0.1(AD1)

P0.2(AD2)

P0.3(AD3)

P0.4(AD4)

P0.5(AD5)

P0.6(AD6)

P0.7(AD7)

EA/VPP

ALE/PROG

PSEN

P2.7(A15)

P2.6(A14)

P2.5(A13)

P2.4(A12)

P2.3(A11)

P2.2(A10)

P2.1(A9)

P2.0(A8)

P0.0(AD0)

P0.1(AD1)

P0.2(AD2)

P0.3(AD3)

P0.4(AD4)

P0.5(AD5)

P0.6(AD6)

P0.7(AD7)

EA/VPP

ALE/PROG

PSEN

P2.7(A15)

P2.6(A14)

P2.5(A13)

P2.4(A12)

P2.3(A11)

P2.2(A10)

P2.1(A9)

P2.0(A8)

P0.0(AD0)

P0.1(AD1)

P0.2(AD2)

P0.3(AD3)

P0.4(AD4)

P0.5(AD5)

P0.6(AD6)

P0.7(AD7)

EA/VPP

ALE/PROG

PSEN

P2.7(A15)

P2.6(A14)

P2.5(A13)

P2.4(A12)

P2.3(A11)

P2.2(A10)

P2.1(A9)

P2.0(A8)

P0.0(AD0)

P0.1(AD1)

P0.2(AD2)

P0.3(AD3)

P0.4(AD4)

P0.5(AD5)

P0.6(AD6)

P0.7(AD7)

EA/VPP

ALE/PROG

PSEN

P2.7(A15)

P2.6(A14)

P2.5(A13)

P2.4(A12)

P2.3(A11)

P2.2(A10)

P2.1(A9)

P2.0(A8)
Review Questions:

1. What is a state in 8031/8051 microcontroller?
2. How many machine cycles are needed to execute an instruction in 8031 / 8051 controller?
3. How to estimate the time taken to execute an instruction in 8031/8051 controller?
4. What is the size of 8031/8051 instructions?
5. List the various machine cycles of 8031/8051 controller.
6. How the 8051 microcontroller differentiates external program memory access and data memory access?
7. What are the addressing modes available in 8051 Controller?
8. Explain the register indirect addressing in 8051.
9. Explain the relative addressing in 8051.
10. How the 8051 instructions can be classified?
11. List the instructions of 8051 that affect all the flags of 8051.
12. List the instructions of 8051 that affect overflow flag in 8051.
13. List the instructions of 8051 that affect only carry flag.
14. List the instructions of 8051 that always clear carry flag.
15. What are the operations performed by Boolean variable instructions of 8051?
8 BIT DATA ADDITION

AIM:

To add two 8 bit numbers stored at consecutive memory locations.

ALGORITHM:

1. Initialize memory pointer to data location.
2. Get the first number from memory in accumulator.
3. Get the second number and add it to the accumulator.
4. Store the answer at another memory location.

OBSERVATION:

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>4500</td>
<td>4502</td>
</tr>
<tr>
<td>4501</td>
<td>4503</td>
</tr>
</tbody>
</table>

RESULT:
Thus the 8 bit numbers stored at 4500 & 4501 are added and the result stored at 4502 & 4503.
START

[C] ← 00H

[HL] ← 4500H

[A] ← [M]

[HL] ← [HL] + 1

[A] ← [A] + [M]

Is there a Carry?

[C] ← [C] + 1

[HL] ← [HL] + 1

[M] ← [A]

[HL] ← [HL] + 1

[M] ← [C]

STOP
8 BIT DATA SUBTRACTION

AIM:
To Subtract two 8 bit numbers stored at consecutive memory locations.

ALGORITHM:

1. Initialize memory pointer to data location.
2. Get the first number from memory in accumulator.
3. Get the second number and subtract from the accumulator.
4. If the result yields a borrow, the content of the acc. is complemented and 01H is added to it (2’s complement). A register is cleared and the content of that reg. is incremented in case there is a borrow. If there is no borrow the content of the acc. is directly taken as the result.
5. Store the answer at next memory location.

OBSERVATION:

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>4500</td>
<td>4502</td>
</tr>
<tr>
<td>4501</td>
<td>4503</td>
</tr>
</tbody>
</table>

RESULT:
Thus the 8 bit numbers stored at 4500 & 4501 are subtracted and the result stored at 4502 & 4503.
FLOW CHART:

START

[C] ← 00H

[HL] ← 4500H

[A] ← [M]

[HL] ← [HL]+1

[A] ← [A]-[M]

Is there a Borrow?

NO

YES

Complement [A]
Add 01H to [A]

[C] ← [C]+1

[HL] ← [HL]+1

[M] ← [A]

[HL] ← [HL]+1

[M] ← [C]

STOP
8 BIT DATA MULTIPLICATION

AIM:
To multiply two 8 bit numbers stored at consecutive memory locations and store the result in memory.

ALGORITHM:

LOGIC: Multiplication can be done by repeated addition.

1. Initialize memory pointer to data location.
2. Move multiplicand to a register.
3. Move the multiplier to another register.
4. Clear the accumulator.
5. Add multiplicand to accumulator
6. Decrement multiplier
7. Repeat step 5 till multiplier comes to zero.
8. The result, which is in the accumulator, is stored in a memory location.
FLOW CHART:

START

[HL] ← 4500

B ← M

[HL] ← [HL]+1

A ← 00

C ← 00

[A] ← [A] + [M]

Is there any carry

YES

C ← C+1

B ← B-1

IS B=0

YES

A
**OBSERVATION:**

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>4500</td>
<td>4502</td>
</tr>
<tr>
<td>4501</td>
<td>4503</td>
</tr>
</tbody>
</table>

**RESULT:**

Thus the 8-bit multiplication was done in 8085µp using repeated addition method.
8 BIT DIVISION

AIM:

To divide two 8-bit numbers and store the result in memory.

ALGORITHM:

LOGIC: Division is done using the method Repeated subtraction.
1. Load Divisor and Dividend
2. Subtract divisor from dividend
3. Count the number of times of subtraction which equals the quotient
4. Stop subtraction when the dividend is less than the divisor. The dividend now becomes the remainder. Otherwise go to step 2.
5. Stop the program execution.

OBSERVATION:

<table>
<thead>
<tr>
<th>S.NO</th>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ADDRESS</td>
<td>DATA</td>
</tr>
<tr>
<td>1</td>
<td>4500</td>
<td>4502</td>
</tr>
<tr>
<td></td>
<td>4501</td>
<td>4503</td>
</tr>
<tr>
<td>2</td>
<td>4500</td>
<td>4502</td>
</tr>
<tr>
<td></td>
<td>4501</td>
<td>4503</td>
</tr>
</tbody>
</table>

RESULT:

Thus an ALP was written for 8-bit division using repeated subtraction method and executed using 8085µ p kits
FLOWCHART:

START

B ← 00

[HL] ← 4500

A ← M

[HL] ← [HL]+1

M ← A-M

[B] ← [B]+1

IS A<0

YES

A ← A+ M

B ← B-1

[HL] ← [HL]+1

NO

[HL] ← [HL]+1

[HL] ← [HL]+1

[HL] ← [HL]+1

[M] ← [A]

[M] ← [B]

STOP
16 BIT DATA ADDITION

AIM:
To add two 16-bit numbers stored at consecutive memory locations.

ALGORITHM:

1. Initialize memory pointer to data location.
2. Get the first number from memory and store in Register pair.
3. Get the second number in memory and add it to the Register pair.
4. Store the sum & carry in separate memory locations.

OBSERVATION:

<table>
<thead>
<tr>
<th>INPUT ADDRESS</th>
<th>DATA</th>
<th>OUTPUT ADDRESS</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>8050H</td>
<td></td>
<td>8054H</td>
<td></td>
</tr>
<tr>
<td>8051H</td>
<td></td>
<td>8055H</td>
<td></td>
</tr>
<tr>
<td>8052H</td>
<td></td>
<td>8056H</td>
<td></td>
</tr>
<tr>
<td>8053H</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RESULT:
Thus an ALP program for 16-bit addition was written and executed in 8085μp using special instructions.
FLOW CHART:

START

[L] ← [8050 H]
[H] ← [8051 H]

[DE] ← [HL]

[L] ← [8052H]
[H] ← [8053H]

[A] ← 00H

[HL] ← [HL] + [DE]

Is there a Carry?

NO

YES

[A] ← [A] + 1

[8054] ← [L]

[8055] ← [H]

[8056] ← [A]

STOP
16 BIT DATA SUBTRACTION

AIM:
To subtract two 16-bit numbers stored at consecutive memory locations.

ALGORITHM:

1. Initialize memory pointer to data location.
2. Get the subtrahend from memory and transfer it to register pair.
3. Get the minuend from memory and store it in another register pair.
4. Subtract subtrahend from minuend.
5. Store the difference and borrow in different memory locations.

OBSERVATION:

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDRESS</td>
<td>DATA</td>
</tr>
<tr>
<td>8050H</td>
<td></td>
</tr>
<tr>
<td>8051H</td>
<td></td>
</tr>
<tr>
<td>8052H</td>
<td></td>
</tr>
<tr>
<td>8053H</td>
<td></td>
</tr>
</tbody>
</table>

RESULT: Thus an ALP program for subtracting two 16-bit numbers was written and executed.
16 BIT MULTIPLICATION

AIM:

To multiply two 16 bit numbers and store the result in memory.

ALGORITHM:

1. Get the multiplier and multiplicand.
2. Initialize a register to store partial product.
3. Add multiplicand, multiplier times.
4. Store the result in consecutive memory locations.
FLOWCHART:

START

L ← [8050]
H ← [8051]

SP ← HL

L ← [8052]
H ← [8053]

DE ← HL

HL ← 0000
BC ← 0000

HL ← HL + SP

Is Carry flag set?

YES

BC ← BC + 1

DE ← DE + 1

NO

Is Zero flag set?

YES

A
OBSERVATION:

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDRESS</td>
<td>DATA</td>
</tr>
<tr>
<td>8050</td>
<td></td>
</tr>
<tr>
<td>8051</td>
<td></td>
</tr>
<tr>
<td>8052</td>
<td></td>
</tr>
<tr>
<td>8053</td>
<td></td>
</tr>
</tbody>
</table>

RESULT:

Thus the 16-bit multiplication was done in 8085μp using repeated addition method.
16- BIT DIVISION

AIM:
To divide two 16-bit numbers and store the result in memory using 8085 mnemonics.

ALGORITHM:
1. Get the dividend and divisor.
2. Initialize the register for quotient.
3. Repeatedly subtract divisor from dividend till dividend becomes less than divisor.
4. Count the number of subtraction which equals the quotient.
5. Store the result in memory.
FLOWCHART:

START

L ← [8051]
H ← [8052]

HL ← DE

L ← [8050]
H ← [8051]

BC ← 0000H

A ← L; A ← A - E
L ← A

A ← H
A ← A - H - Borrow
H ← A

BC ← BC + 1

Is Carry flag set?

NO

YES

A
RESULT:
Thus the 16-bit Division was done in 8085μp using repeated subtraction method.
LARGEST ELEMENT IN AN ARRAY

AIM:

To find the largest element in an array.

ALGORITHM:

1. Place all the elements of an array in the consecutive memory locations.
2. Fetch the first element from the memory location and load it in the accumulator.
3. Initialize a counter (register) with the total number of elements in an array.
4. Decrement the counter by 1.
5. Increment the memory pointer to point to the next element.
6. Compare the accumulator content with the memory content (next element).
7. If the accumulator content is smaller, then move the memory content (largest element) to the accumulator. Else continue.
8. Decrement the counter by 1.
9. Repeat steps 5 to 8 until the counter reaches zero
10. Store the result (accumulator content) in the specified memory location.

OBSERVATION:

<table>
<thead>
<tr>
<th>INPUT ADDRESS</th>
<th>INPUT DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>8100</td>
<td></td>
</tr>
<tr>
<td>8101</td>
<td></td>
</tr>
<tr>
<td>8102</td>
<td></td>
</tr>
<tr>
<td>8103</td>
<td></td>
</tr>
<tr>
<td>8104</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OUTPUT ADDRESS</th>
<th>OUTPUT DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>8105</td>
<td></td>
</tr>
</tbody>
</table>

RESULT: Thus the largest number in the given array is found out.
START

[HL] ← [8100H]

[B] ← 04H

[A] ← [HL]

[HL] ← [HL] + 1

IS [A] < [HL]?

YES

[A] ← [HL]

[B] ← [B] - 1

IS [B] = 0?

YES

[8105] ← [A]

STOP

NO
SMALLEST ELEMENT IN AN ARRAY

AIM:
To find the smallest element in an array.

ALGORITHM:
1. Place all the elements of an array in the consecutive memory locations.
2. Fetch the first element from the memory location and load it in the accumulator.
3. Initialize a counter (register) with the total number of elements in an array.
4. Decrement the counter by 1.
5. Increment the memory pointer to point to the next element.
6. Compare the accumulator content with the memory content (next element).
7. If the accumulator content is smaller, then move the memory content (largest element) to the accumulator. Else continue.
8. Decrement the counter by 1.
9. Repeat steps 5 to 8 until the counter reaches zero.
10. Store the result (accumulator content) in the specified memory location.

OBSERVATION:

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDRESS</td>
<td>DATA</td>
</tr>
<tr>
<td>8100</td>
<td></td>
</tr>
<tr>
<td>8101</td>
<td></td>
</tr>
<tr>
<td>8102</td>
<td></td>
</tr>
<tr>
<td>8103</td>
<td></td>
</tr>
<tr>
<td>8104</td>
<td></td>
</tr>
</tbody>
</table>

RESULT: Thus the smallest number in the given array is found out.
FLOW CHART:

START

[HL] ← [8100H]

[B] ← 04H

[A] ← [HL]

[HL] ← [HL] + 1

IS [A] < [HL]?

YES

NO

[A] ← [HL]

[B] ← [B] - 1

IS [B] = 0?

YES

[B] ← [B] - 1

[8105] ← [A]

STOP
ASCENDING ORDER

**AIM:**
To sort the given number in the ascending order using 8085 microprocessor.

**ALGORITHM:**

1. Get the numbers to be sorted from the memory locations.
2. Compare the first two numbers and if the first number is larger than second then I interchange the number.
3. If the first number is smaller, go to step 4
4. Repeat steps 2 and 3 until the numbers are in required order
FLOWCHART:

START

[B] ← 04H

[HL] ← [8100H]

[C] ← 04H

[A] ← [HL]

[HL] ← [HL] + 1

IS [A] < [HL]?

YES

[D] ← [HL]

[HL] ← [A]

[HL] ← [HL] - 1

[HL] ← [D]

[HL] ← [HL] + 1

[C] ← [C] - 01H

A

C

B
RESULT:
Thus the ascending order program is executed and thus the numbers are arranged in ascending order.

<table>
<thead>
<tr>
<th>INPUT MEMORY LOCATION</th>
<th>DATA</th>
<th>OUTPUT MEMORY LOCATION</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>8100</td>
<td></td>
<td>8100</td>
<td></td>
</tr>
<tr>
<td>8101</td>
<td></td>
<td>8101</td>
<td></td>
</tr>
<tr>
<td>8102</td>
<td></td>
<td>8102</td>
<td></td>
</tr>
<tr>
<td>8103</td>
<td></td>
<td>8103</td>
<td></td>
</tr>
<tr>
<td>8104</td>
<td></td>
<td>8104</td>
<td></td>
</tr>
</tbody>
</table>
DESCENDING ORDER

AIM:
To sort the given number in the descending order using 8085 microprocessor.

ALGORITHM:

1. Get the numbers to be sorted from the memory locations.
2. Compare the first two numbers and if the first number is smaller than second then interchange the number.
3. If the first number is larger, go to step 4
4. Repeat steps 2 and 3 until the numbers are in required order
FLOWCHART:

START

[B] ← 04H

[HL] ← [8100H]

[C] ← 04H

[A] ← [HL]

[HL] ← [HL] + 1

IS [A] < [HL]?

NO

YES

[D] ← [HL]

[HL] ← [A]

[HL] ← [HL] - 1

[HL] ← [D]

[HL] ← [HL] + 1

[C] ← [C] - 01H

A

B

C
RESULT:
Thus the descending order program is executed and thus the numbers are arranged in descending order.
CODE CONVERSION – DECIMAL TO HEX

AIM:

To convert a given decimal number to hexadecimal.

ALGORITHM:

1. Initialize the memory location to the data pointer.
2. Increment B register.
3. Increment accumulator by 1 and adjust it to decimal every time.
4. Compare the given decimal number with accumulator value.
5. When both matches, the equivalent hexadecimal value is in B register.
6. Store the resultant in memory location.

OBSERVATION:

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDRESS</td>
<td>DATA</td>
</tr>
<tr>
<td>8100</td>
<td></td>
</tr>
</tbody>
</table>

RESULT:

Thus an ALP program for conversion of decimal to hexadecimal was written and executed.
FLOWCHART:

START

HL ← 4500H

A ← 00

B ← 00H

B ← B+1

A ← A +1

Decimal adjust accumulator

Is A=M?

NO

YES

A ← B

8101 ← A

Stop
CODE CONVERSION –HEXADECIMAL TO DECIMAL

AIM:
To convert a given hexadecimal number to decimal.

ALGORITHM:
1. Initialize the memory location to the data pointer.
2. Increment B register.
3. Increment accumulator by 1 and adjust it to decimal every time.
4. Compare the given hexadecimal number with B register value.
5. When both match, the equivalent decimal value is in A register.
6. Store the resultant in memory location.

OBSERVATION:

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDRESS DATA</td>
<td>ADDRESS DATA</td>
</tr>
<tr>
<td>8100</td>
<td>8101</td>
</tr>
<tr>
<td></td>
<td>8102</td>
</tr>
</tbody>
</table>

RESULT:
Thus an ALP program for conversion of hexadecimal to decimal was written and executed.
FLOWCHART:

START

HL ← 8100H

A ← 00

B ← 00H

C ← 00H

B ← B+1

A ← A +1

Decimal adjust accumulator

Is there carry?

C ← C+1

D ← A, A ← B,

Is A=M?

NO

8101 ← A, A ← C
8102 ← A

Stop
**BCD ADDITION**

**AIM:**
To add two 8 bit BCD numbers stored at consecutive memory locations.

**ALGORITHM:**
1. Initialize memory pointer to data location.
2. Get the first number from memory in accumulator.
3. Get the second number and add it to the accumulator
4. Adjust the accumulator value to the proper BCD value using DAA instruction.
5. Store the answer at another memory location.

**OBSERVATION:**

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>4500</td>
<td>4502</td>
</tr>
<tr>
<td>4501</td>
<td>4503</td>
</tr>
</tbody>
</table>

**RESULT:**
Thus the 8 bit BCD numbers stored at 4500 & 4501 are added and the result stored at 4502 & 4503.
FLOW CHART:

START

[C] ← 00H

[HL] ← 4500H

[A] ← [M]

[HL] ← [HL]+1

[A] ← [A]+[M]

Decimal Adjust Accumulator

Is there a Carry?

[C] ← [C]+1

[HL] ← [HL]+1

[HL] ← [HL]+1

[M] ← [A]

[M] ← [C]

STOP
BCD SUBTRACTION

AIM:
To Subtract two 8 bit BCD numbers stored at consecutive memory locations.

ALGORITHM:
1. Load the minuend and subtrahend in two registers.
2. Initialize Borrow register to 0.
3. Take the 100’s complement of the subtrahend.
4. Add the result with the minuend which yields the result.
5. Adjust the accumulator value to the proper BCD value using DAA instruction. If there is a carry ignore it.
6. If there is no carry, increment the carry register by 1
7. Store the content of the accumulator (result) and borrow register in the specified memory location

OBSERVATION:

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>4500</td>
<td>4502</td>
</tr>
<tr>
<td>4501</td>
<td>4503</td>
</tr>
</tbody>
</table>

RESULT:
Thus the 8 bit BCD numbers stored at 4500 & 4501 are subtracted and the result stored at 4502 & 4503.
FLOW CHART:

START

[D] ← 00H
HL ← 4500
B ← M

HL ← HL + 1
C ← M
A ← 99

[A] ← [A] + [B]
DAA

Is there a Carry?

YES

NO

[D] ← [D] + 1

[HL] ← [HL] + 1

[4502] ← A
[4503] ← D

STOP
BIOS/DOS CALLS – DISPLAY

AIM:
To display a message on the CRT screen of a microcomputer using DOS calls.

ALGORITHM:
1. Initialize the data segment and the message to be displayed.
2. Set function value for display.
3. Point to the message and run the interrupt to display the message in the CRT.

PROGRAM:
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
MSG DB 0DH, 0AH, "GOOD MORNING" , 0DH, 0AH, "$"
DATA ENDS
CODE SEGMENT
START:   MOV AX, DATA
          MOV DS, AX
          MOV AH, 09H
          MOV DX, OFFSET MSG
          INT 21H
          MOV AH, 4CH
          INT 21H
CODE ENDS
END START

RESULT:
A message is displayed on the CRT screen of a microcomputer using DOS calls.
BIOS/DOS CALLS – FILE MANIPULATION

**AIM:** To open a file using DOS calls.

**ALGORITHM:**
1. Initialize the data segment, file name and the message to be displayed.
2. Set the file attribute to create a file using a DOS call.
3. If the file is unable to create a file display the message

**PROGRAM:**

```assembly
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
FILENAME DB "SAMPLE.DAT", "$"
MSG DB 0DH, 0AH, "FILE NOT CREATED", 0DH, 0AH, "$"
DATA ENDS
CODE SEGMENT
START: MOV AX, DATA
        MOV DS, AX
        MOV DX, OFFSET FILENAME
        MOV CX, 00H
        MOV AH, 3CH
        INT 21H
        JNC LOOP1
        MOV AX, DATA
        MOV DS, AX
        MOV DX, OFFSET MSG
        MOV AH, 09H
        INT 21H
LOOP1:  MOV AH, 4CH
        INT 21H
CODE ENDS
END START
```

**RESULT:** A file is opened using DOS calls.
BIOS/DOS CALLS – DISK INFORMATION

AIM:
To display the disk information.

ALGORITHM:
1. Initialize the data segment and the message to be displayed.
2. Set function value for disk information.
3. Point to the message and run the interrupt to display the message in the CRT.

PROGRAM:

ASSUME CS: CODE, DS: DATA
DATA SEGMENT
MSG DB 0DH, 0AH, “GOOD MORNING” , ODH, OAH, “$”
DATA ENDS
CODE SEGMENT
START: MOV AX, DATA
       MOV DS, AX
       MOV AH, 36H
       MOV DX, OFFSET MSG
       INT 21H
       MOV AH, 4CH
       INT 21H
CODE ENDS
END START

RESULT:
The disk information is displayed.
8086 STRING MANIPULATION – SEARCH A WORD

**AIM:**
To search a word from a string.

**ALGORITHM:**
1. Load the source and destination index register with starting and the ending address respectively.
2. Initialize the counter with the total number of words to be copied.
3. Clear the direction flag for auto incrementing mode of transfer.
4. Use the string manipulation instruction SCASW with the prefix REP to search a word from string.
5. If a match is found (z=1), display 01 in destination address. Otherwise, display 00 in destination address.
**PROGRAM:**

ASSUME CS: CODE, DS: DATA

DATA SEGMENT
LIST DW 53H, 15H, 19H, 02H
DEST EQU 3000H
COUNT EQU 05H
DATA ENDS

CODE SEGMENT

START:

MOV AX, DATA
MOV DS, AX
MOV AX, 15H
MOV SI, OFFSET LIST
MOV DI, DEST
MOV CX, COUNT
MOV AX, 00
CLD

REP SCASW
JZ LOOP
MOV AX, 01

LOOP MOV [DI], AX
MOV AH, 4CH
INT 21H

CODE ENDS

END START

**INPUT:**

LIST: 53H, 15H, 19H, 02H

**OUTPUT:**

3000 01

**RESULT:**

A word is searched and the count of number of appearances is displayed.
8086 STRING MANIPULATION – FIND AND REPLACE A WORD

**AIM:**
To find and replace a word from a string.

**ALGORITHM:**
1. Load the source and destination index register with starting and the ending address respectively.
2. Initialize the counter with the total number of words to be copied.
3. Clear the direction flag for auto incrementing mode of transfer.
4. Use the string manipulation instruction SCASW with the prefix REP to search a word from string.
5. If a match is found (z=1), replace the old word with the current word in destination address. Otherwise, stop.
** PROGRAM: **
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
LIST DW 53H, 15H, 19H, 02H
REPLACE EQU 30H
COUNT EQU 05H
DATA ENDS
CODE SEGMENT
START: MOV AX, DATA
       MOV DS, AX
       MOV AX, 15H
       MOV SI, OFFSET LIST
       MOV CX, COUNT
       MOV AX, 00
       CLD
       REP SCASW
       JNZ LOOP
       MOV DI, LABEL LIST
       MOV [DI], REPLACE
LOOP MOV AH, 4CH
       INT 21H
CODE ENDS
END START

** INPUT: **
LIST: 53H, 15H, 19H, 02H

** OUTPUT: **
LIST: 53H, 30H, 19H, 02H

** RESULT: **
A word is found and replaced from a string.
AIM:
To copy a string of data words from one location to the other.

ALGORITHM:
1. Load the source and destination index register with starting and the ending address respectively.
2. Initialize the counter with the total number of words to be copied.
3. Clear the direction flag for auto incrementing mode of transfer.
4. Use the string manipulation instruction MOVSW with the prefix REP to copy a string from source to destination.
**PROGRAM:**

```
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
SOURCE EQU 2000H
DEST EQU 3000H
COUNT EQU 05H
DATA ENDS
CODE SEGMENT
START:    MOV AX, DATA
           MOV DS, AX
           MOV ES, AX
           MOV SI, SOURCE
           MOV DI, DEST
           MOV CX, COUNT
           CLD
           REP MOVSW
           MOV AH, 4CH
           INT 21H
CODE ENDS
END START
```

**INPUT:**

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>48</td>
</tr>
<tr>
<td>2001</td>
<td>84</td>
</tr>
<tr>
<td>2002</td>
<td>67</td>
</tr>
<tr>
<td>2003</td>
<td>90</td>
</tr>
<tr>
<td>2004</td>
<td>21</td>
</tr>
</tbody>
</table>

**OUTPUT:**

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>3000</td>
<td>48</td>
</tr>
<tr>
<td>3001</td>
<td>84</td>
</tr>
<tr>
<td>3002</td>
<td>67</td>
</tr>
<tr>
<td>3003</td>
<td>90</td>
</tr>
<tr>
<td>3004</td>
<td>21</td>
</tr>
</tbody>
</table>

**RESULT:**

A string of data words is copied from one location to other.
AIM:
To sort a group of data bytes.

ALGORITHM:
• Place all the elements of an array named list (in the consecutive memory locations).
• Initialize two counters DX & CX with the total number of elements in the array.
• Do the following steps until the counter B reaches 0.
  o Load the first element in the accumulator
  o Do the following steps until the counter C reaches 0.
    1. Compare the accumulator content with the next element present in the next memory location. If the accumulator content is smaller go to next step; otherwise, swap the content of accumulator with the content of memory location.
    2. Increment the memory pointer to point to the next element.
    3. Decrement the counter C by 1.
• Stop the execution.
**PROGRAM:**

```
ASSUME CS: CODE, DS: DATA

DATA SEGMENT
LIST DW 53H, 25H, 19H, 02H
COUNT EQU 04H
DATA ENDS

CODE SEGMENT
START:     MOV AX, DATA
           MOV DS, AX
           MOV DX, COUNT-1
LOOP2:     MOV CX, DX
           MOV SI, OFFSET LIST
AGAIN:     MOV AX, [SI]
           CMP AX, [SI+2]
           JC LOOP1
           XCHG [SI +2], AX
           XCHG [SI], AX
LOOP1:     ADD SI, 02
           LOOP AGAIN
           DEC DX
           JNZ LOOP2
           MOV AH, 4CH
           INT 21H

CODE ENDS
END START
```

**INPUT:**

LIST: 53H, 25H, 19H, 02H

**OUTPUT:**

LIST: 02H, 19H, 25H, 53H

**RESULT:**

A group of data bytes are arranged in ascending order.
INTERFACING 8255 WITH 8085

AIM:
To interface programmable peripheral interface 8255 with 8085 and study its characteristics in mode0, mode1 and BSR mode.

APPARATUS REQUIRED:
8085 μp kit, 8255 interface board, DC regulated power supply, VXT parallel bus

I/O MODES:

Control Word:
MODE 0 – SIMPLE I/O MODE:

This mode provides simple I/O operations for each of the three ports and is suitable for synchronous data transfer. In this mode all the ports can be configured either as input or output port.

Let us initialize port A as input port and port B as output port

PROGRAM:

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>OPCODES</th>
<th>LABEL</th>
<th>MNEMONICS</th>
<th>OPERAND</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td></td>
<td>START:</td>
<td>MVI</td>
<td>A, 90</td>
<td>Initialize port A as input and Port B as output.</td>
</tr>
<tr>
<td>4101</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4102</td>
<td></td>
<td>OUT</td>
<td>C6</td>
<td></td>
<td>Send Mode Control word</td>
</tr>
<tr>
<td>4103</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4104</td>
<td></td>
<td>IN</td>
<td>C0</td>
<td></td>
<td>Read from Port A</td>
</tr>
<tr>
<td>4105</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4106</td>
<td></td>
<td>OUT</td>
<td>C2</td>
<td></td>
<td>Display the data in port B</td>
</tr>
<tr>
<td>4107</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4108</td>
<td></td>
<td>STA</td>
<td>4200</td>
<td></td>
<td>Store the data read from Port A in 4200</td>
</tr>
<tr>
<td>4109</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>410A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>410B</td>
<td></td>
<td>HLT</td>
<td></td>
<td></td>
<td>Stop the program.</td>
</tr>
</tbody>
</table>

MODE 1 STROBED I/O MODE:

In this mode, port A and port B are used as data ports and port C is used as control signals for strobed I/O data transfer.

Let us initialize port A as input port in mode 1
### MAIN PROGRAM:

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>OTPCODES</th>
<th>LABEL</th>
<th>MNEMONICS</th>
<th>OPERAND</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>START:</td>
<td>MVI</td>
<td>A, B4</td>
<td></td>
<td>Initialize port A as Input port in mode 1.</td>
</tr>
<tr>
<td>4101</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4102</td>
<td>OUT</td>
<td>C6</td>
<td></td>
<td></td>
<td>Send Mode Control word</td>
</tr>
<tr>
<td>4103</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4104</td>
<td>MVI</td>
<td>A,09</td>
<td></td>
<td></td>
<td>Set the PC4 bit for INTE A</td>
</tr>
<tr>
<td>4105</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4106</td>
<td>OUT</td>
<td>C6</td>
<td></td>
<td></td>
<td>Display the data in port B</td>
</tr>
<tr>
<td>4107</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4108</td>
<td>EI</td>
<td>A,08</td>
<td></td>
<td></td>
<td>Enable RST5.5</td>
</tr>
<tr>
<td>4109</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>410A</td>
<td>SIM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>410B</td>
<td>HLT</td>
<td></td>
<td></td>
<td></td>
<td>Stop the program.</td>
</tr>
</tbody>
</table>

### ISR (Interrupt Service Routine)

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>OTPCODES</th>
<th>LABEL</th>
<th>MNEMONICS</th>
<th>OPERAND</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>4200</td>
<td>START:</td>
<td>IN</td>
<td>C0</td>
<td></td>
<td>Read from port A</td>
</tr>
<tr>
<td>4201</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4202</td>
<td>STA</td>
<td>4500</td>
<td></td>
<td></td>
<td>Store in 4500.</td>
</tr>
<tr>
<td>4203</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4204</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4205</td>
<td>HLT</td>
<td></td>
<td></td>
<td></td>
<td>Stop the program.</td>
</tr>
</tbody>
</table>

### Sub program:

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>OTPCODES</th>
<th>LABEL</th>
<th>MNEMONICS</th>
<th>OPERAND</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>405E</td>
<td>JMP</td>
<td></td>
<td>4200</td>
<td></td>
<td>Go to 4200</td>
</tr>
<tr>
<td>405F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4060</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Any lines of port c can be set or reset individually without affecting other lines using this mode. Let us set PC0 and PC3 bits using this mode.

**PROGRAM:**

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>OPCODES</th>
<th>LABEL</th>
<th>MNEMONICS</th>
<th>OPERAND</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td></td>
<td>START:</td>
<td>MVI</td>
<td>A, 01</td>
<td>Set PC0</td>
</tr>
<tr>
<td>4101</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4102</td>
<td></td>
<td>OUT</td>
<td>C6</td>
<td></td>
<td>Send Mode Control word</td>
</tr>
<tr>
<td>4103</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4104</td>
<td></td>
<td>MVI</td>
<td>A, 07</td>
<td></td>
<td>Set PC3</td>
</tr>
<tr>
<td>4105</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4106</td>
<td></td>
<td>OUT</td>
<td>C6</td>
<td></td>
<td>Send Mode Control word</td>
</tr>
<tr>
<td>4107</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4109</td>
<td></td>
<td>HLT</td>
<td></td>
<td></td>
<td>Stop the program.</td>
</tr>
</tbody>
</table>

**RESULT:**

Thus 8255 is interfaced and its characteristics in mode0, mode1 and BSR mode is studied.
INTERFACING 8253 TIMER WITH 8085

Interfacing 8253 Programmable Interval Timer with 8085 µp

AIM:
To interface 8253 Interface board to 8085 µp and verify the operation of 8253 in six different modes.

APPARATUS REQUIRED:
8085 µp kit, 8253 Interface board, DC regulated power supply, VXT parallel bus, CRO.

Mode 0 – Interrupt on terminal count:
The output will be initially low after mode set operations. After loading the counter, the output will be remaining low while counting and on terminal count; the output will become high, until reloaded again.

Let us set the channel 0 in mode 0. Connect the CLK 0 to the debounce circuit by changing the jumper J3 and then execute the following program.

Program:

<table>
<thead>
<tr>
<th>Address</th>
<th>Opcodes</th>
<th>Label</th>
<th>Mnemonic</th>
<th>Operands</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>START:</td>
<td>MVI</td>
<td>A, 30</td>
<td></td>
<td>Channel 0 in mode 0</td>
</tr>
<tr>
<td>4102</td>
<td>OUT</td>
<td>CE</td>
<td></td>
<td></td>
<td>Send Mode Control word</td>
</tr>
<tr>
<td>4104</td>
<td>MVI</td>
<td>A, 05</td>
<td></td>
<td></td>
<td>LSB of count</td>
</tr>
<tr>
<td>4106</td>
<td>OUT</td>
<td>C8</td>
<td></td>
<td></td>
<td>Write count to register</td>
</tr>
<tr>
<td>4108</td>
<td>MVI</td>
<td>A, 00</td>
<td></td>
<td></td>
<td>MSB of count</td>
</tr>
<tr>
<td>410A</td>
<td>OUT</td>
<td>C8</td>
<td></td>
<td></td>
<td>Write count to register</td>
</tr>
<tr>
<td>410C</td>
<td>HLT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

It is observed in CRO that the output of Channel 0 is initially LOW. After giving six clock pulses, the output goes HIGH.
**Mode 1 – Programmable ONE-SHOT:**

After loading the counter, the output will remain low following the rising edge of the gate input. The output will go high on the terminal count. It is retriggerable; hence the output will remain low for the full count, after any rising edge of the gate input.

**Example:**

The following program initializes channel 0 of 8253 in Mode 1 and also initiates triggering of Gate 0. OUT 0 goes low, as clock pulse after triggering the goes back to high level after 5 clock pulses. Execute the program, give clock pulses through the debounce logic and verify using CRO.

<table>
<thead>
<tr>
<th>Address</th>
<th>Opcodes</th>
<th>Label</th>
<th>Mnemonic</th>
<th>Operands</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>START:</td>
<td>MVI</td>
<td>A, 32</td>
<td></td>
<td>Channel 0 in mode 1</td>
</tr>
<tr>
<td>4102</td>
<td>OUT</td>
<td>CE</td>
<td></td>
<td></td>
<td>Send Mode Control word</td>
</tr>
<tr>
<td>4104</td>
<td>MVI</td>
<td>A, 05</td>
<td></td>
<td>LSB of count</td>
<td></td>
</tr>
<tr>
<td>4106</td>
<td>OUT</td>
<td>C8</td>
<td></td>
<td></td>
<td>Write count to register</td>
</tr>
<tr>
<td>4108</td>
<td>MVI</td>
<td>A, 00</td>
<td></td>
<td>MSB of count</td>
<td></td>
</tr>
<tr>
<td>410A</td>
<td>OUT</td>
<td>C8</td>
<td></td>
<td></td>
<td>Write count to register</td>
</tr>
<tr>
<td>410C</td>
<td>OUT</td>
<td>D0</td>
<td></td>
<td>Trigger Gate0</td>
<td></td>
</tr>
<tr>
<td>4100</td>
<td>HLT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Mode 2 – Rate Generator:**

It is a simple divide by N counter. The output will be low for one period of the input clock. The period from one output pulse to the next equals the number of input counts in the count register. If the count register is reloaded between output pulses the present period will not be affected but the subsequent period will reflect the new value.
Example:

Using Mode 2, let us divide the clock present at Channel 1 by 10. Connect the CLK1 to PCLK.

<table>
<thead>
<tr>
<th>Address</th>
<th>Opcodes</th>
<th>Label</th>
<th>Mnemonic</th>
<th>Operands</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>3E 74</td>
<td>START:</td>
<td>MVI</td>
<td>A, 74</td>
<td>Channel 1 in mode 2</td>
</tr>
<tr>
<td>4102</td>
<td>D3 CE</td>
<td></td>
<td>OUT</td>
<td>CE</td>
<td>Send Mode Control word</td>
</tr>
<tr>
<td>4104</td>
<td>3E 0A</td>
<td></td>
<td>MVI</td>
<td>A, 0A</td>
<td>LSB of count</td>
</tr>
<tr>
<td>4106</td>
<td>D3 CA</td>
<td></td>
<td>OUT</td>
<td>CA</td>
<td>Write count to register</td>
</tr>
<tr>
<td>4108</td>
<td>3E 00</td>
<td></td>
<td>MVI</td>
<td>A, 00</td>
<td>MSB of count</td>
</tr>
<tr>
<td>410A</td>
<td>D3 CA</td>
<td></td>
<td>OUT</td>
<td>CA</td>
<td>Write count to register</td>
</tr>
<tr>
<td>410C</td>
<td>76</td>
<td></td>
<td>HLT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In CRO observe simultaneously the input clock to channel 1 and the output at Out1.

**Mode 3 Square wave generator:**

It is similar to Mode 2 except that the output will remain high until one half of count and go low for the other half for even number count. If the count is odd, the output will be high for \((\text{count} + 1)/2\) counts. This mode is used of generating Baud rate for 8251A (USART).

Example:

We utilize Mode 0 to generate a square wave of frequency 150 KHz at channel 0.

<table>
<thead>
<tr>
<th>Address</th>
<th>Opcodes</th>
<th>Label</th>
<th>Mnemonic</th>
<th>Operands</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>3E 36</td>
<td>START:</td>
<td>MVI</td>
<td>A, 36</td>
<td>Channel 0 in mode 3</td>
</tr>
<tr>
<td>4102</td>
<td>D3 CE</td>
<td></td>
<td>OUT</td>
<td>CE</td>
<td>Send Mode Control word</td>
</tr>
<tr>
<td>4104</td>
<td>3E 0A</td>
<td></td>
<td>MVI</td>
<td>A, 0A</td>
<td>LSB of count</td>
</tr>
<tr>
<td>4106</td>
<td>D3 C8</td>
<td></td>
<td>OUT</td>
<td>C8</td>
<td>Write count to register</td>
</tr>
<tr>
<td>4108</td>
<td>3E 00</td>
<td></td>
<td>MVI</td>
<td>A, 00</td>
<td>MSB of count</td>
</tr>
<tr>
<td>410A</td>
<td>D3 C8</td>
<td></td>
<td>OUT</td>
<td>C8</td>
<td>Write count to register</td>
</tr>
<tr>
<td>410C</td>
<td>76</td>
<td></td>
<td>HLT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Set the jumper, so that the clock 0 of 8253 is given a square wave of frequency 1.5 MHz. This program divides this PCLK by 10 and thus the output at channel 0 is 150 KHz.

Vary the frequency by varying the count. Here the maximum count is FFFF H. So, the square wave will remain high for 7FFF H counts and remain low for 7FFF H counts. Thus with the input clock frequency of 1.5 MHz, which corresponds to a period of 0.067 microseconds, the resulting square wave has an ON time of 0.02184 microseconds and an OFF time of 0.02184 microseconds.

To increase the time period of square wave, set the jumpers such that CLK2 of 8253 is connected to OUT 0. Using the above-mentioned program, output a square wave of frequency 150 KHz at channel 0. Now this is the clock to channel 2.

**Mode 4: Software Triggered Strobe:**

The output is high after mode is set and also during counting. On terminal count, the output will go low for one clock period and becomes high again. This mode can be used for interrupt generation.

The following program initializes channel 2 of 8253 in mode 4.

**Example:**

Connect OUT 0 to CLK 2 (jumper J1). Execute the program and observe the output OUT 2. Counter 2 will generate a pulse after 1 second.
<table>
<thead>
<tr>
<th>Address</th>
<th>Opcodes</th>
<th>Label</th>
<th>Mnemonic</th>
<th>Operands</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>START:</td>
<td></td>
<td>MVI</td>
<td>A, 36</td>
<td>Channel 0 in mode 0</td>
</tr>
<tr>
<td>4102</td>
<td></td>
<td></td>
<td>OUT</td>
<td>CE</td>
<td>Send Mode Control word</td>
</tr>
<tr>
<td>4104</td>
<td></td>
<td></td>
<td>MVI</td>
<td>A, 0A</td>
<td>LSB of count</td>
</tr>
<tr>
<td>4106</td>
<td></td>
<td></td>
<td>OUT</td>
<td>C8</td>
<td>Write count to register</td>
</tr>
<tr>
<td>4108</td>
<td></td>
<td></td>
<td>MVI</td>
<td>A, 00</td>
<td>MSB of count</td>
</tr>
<tr>
<td>410A</td>
<td></td>
<td></td>
<td>OUT</td>
<td>C8</td>
<td>Write count to register</td>
</tr>
<tr>
<td>410C</td>
<td></td>
<td></td>
<td>MVI</td>
<td>A, B8</td>
<td>Channel 2 in Mode 4</td>
</tr>
<tr>
<td>410E</td>
<td></td>
<td></td>
<td>OUT</td>
<td>CE</td>
<td>Send Mode control Word</td>
</tr>
<tr>
<td>4110</td>
<td></td>
<td></td>
<td>MVI</td>
<td>A, 98</td>
<td>LSB of Count</td>
</tr>
<tr>
<td>4112</td>
<td></td>
<td></td>
<td>OUT</td>
<td>CC</td>
<td>Write Count to register</td>
</tr>
<tr>
<td>4114</td>
<td></td>
<td></td>
<td>MVI</td>
<td>A, 3A</td>
<td>MSB of Count</td>
</tr>
<tr>
<td>4116</td>
<td></td>
<td></td>
<td>OUT</td>
<td>CC</td>
<td>Write Count to register</td>
</tr>
<tr>
<td>4118</td>
<td></td>
<td></td>
<td>HLT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Mode 5 Hardware triggered strobe:** Counter starts counting after rising edge of trigger input and output goes low for one clock period when terminal count is reached. The counter is retriggerable.

**Example:**

The program that follows initializes channel 0 in mode 5 and also triggers Gate 0. Connect CLK 0 to debounce circuit.

Execute the program. After giving Six clock pulses, you can see using CRO, the initially HIGH output goes LOW. The output ( OUT 0 pin) goes high on the next clock pulse.

<table>
<thead>
<tr>
<th>Address</th>
<th>Opcodes</th>
<th>Label</th>
<th>Mnemonic</th>
<th>Operands</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>START:</td>
<td></td>
<td>MVI</td>
<td>A, 1A</td>
<td>Channel 0 in mode 5</td>
</tr>
<tr>
<td>4102</td>
<td></td>
<td></td>
<td>OUT</td>
<td>CE</td>
<td>Send Mode Control word</td>
</tr>
<tr>
<td>4104</td>
<td></td>
<td></td>
<td>MVI</td>
<td>A, 05</td>
<td>LSB of count</td>
</tr>
<tr>
<td>4106</td>
<td></td>
<td></td>
<td>OUT</td>
<td>C8</td>
<td>Write count to register</td>
</tr>
<tr>
<td>4108</td>
<td></td>
<td></td>
<td>MVI</td>
<td>A, 00</td>
<td>MSB of count</td>
</tr>
<tr>
<td>410A</td>
<td></td>
<td></td>
<td>OUT</td>
<td>D0</td>
<td>Trigger Gate 0</td>
</tr>
<tr>
<td>410C</td>
<td></td>
<td></td>
<td>HLT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Result:** Thus the 8253 has been interfaced to 8085 μp and six different modes of 8253 have been studied.
INTERFACING 8279 KEYBOARD / DISPLAY CONTROLLER WITH 8085 MICROPROCESSOR

AIM:
To interface 8279 Programmable Keyboard Display Controller to 8085 Microprocessor.

APPARATUS REQUIRED:
1. 8085 Microprocessor tool kit.
2. 8279 Interface board.
3. VXT parallel bus.
4. Regulated D.C power supply.

PROGRAM:
START: LXI H, 4130 H
MVI D, 0F H ; Initialize counter
MVI A, 10 H
OUT C2 H ; Set Mode and Display
MVI A, CC H ; Clear display
OUT C2 H
MVI A, 90 H ; Write Display
OUT C2 H
LOOP: MOV A, M
OUT C0 H
CALL DELAY
INX H
DCR D
JNZ LOOP
JMP START

DELAY: MVI B, A0 H
LOOP2: MVI C, FF H
LOOP1: DCR C
JNZ LOOP1
DCR B
JNZ LOOP@
RET
Pointer equal to 4130. FF repeated eight times.

4130 -FF
4131 -FF
4132 -FF
4133 -FF
4134 -FF
4135 -FF
4136 -FF
4137 -FF
4138 -98
4139 -68
413A -7C
413B -C8
413C -1C
413D -29
413E -FF
413F -FF

RESULT:

Thus 8279 controller was interfaced with 8085 and program for rolling display was executed successfully.
INTERFACING 8251 (USART) WITH 8085 PROCESSOR

AIM: To write a program to initiate 8251 and to check the transmission and reception of character.

THEORY: The 8251 is used as a peripheral device for serial communication and is programmed by the CPU to operate using virtually any serial data transmission technique. The USART accepts data characters from the CPU in parallel format and then converts them into a continuous serial data stream for transmission. Simultaneously, it can receive serial data streams and convert them into parallel data characters for the CPU. The CPU can read the status of USART at any time. These include data transmission errors and control signals.

Prior to starting data transmission or reception, the 8251 must be loaded with a set of control words generated by the CPU. These control signals define the complete functional definition of the 8251 and must immediately follow a RESET operation. Control words should be written into the control register of 8251. These control words are split into two formats:

1. MODE INSTRUCTION WORD
2. COMMAND INSTRUCTION WORD

1. MODE INSTRUCTION WORD

This format defines the Baud rate, Character length, Parity and Stop bits required to work with asynchronous data communication. By selecting the appropriate baud factor sync mode, the 8251 can be operated in Synchronous mode.

Initializing 8251 using the mode instruction to the following conditions.

- 8 Bit data
- No Parity
- Baud rate Factor (16X)
- 1 Stop Bit gives a mode command word of 0100 1110 = 4E (HEX)
MODE INSTRUCTION - SYNCHRONOUS MODE

<table>
<thead>
<tr>
<th>S2</th>
<th>S1</th>
<th>EP</th>
<th>PEN</th>
<th>L2</th>
<th>L1</th>
<th>B2</th>
<th>B1</th>
</tr>
</thead>
</table>

### BAUD RATE FACTOR

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

SYNC MODE: (1X) (16X) (64X)

### CHARACTER LENGTH

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

5 BITS 6 BITS 7 BITS 8 BITS

### PARITY ENABLE

1 = ENABLE 0 = DISABLE

### EVEN PARITY GEN/CHECK

0 = ODD 1 = EVEN

### NUMBER OF STOP BITS

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

INVALID 1 BIT 1.5 BIT 2 BIT
MODE INSTRUCTION - ASYNCHRONOUS MODE

<table>
<thead>
<tr>
<th>S2</th>
<th>S1</th>
<th>EP</th>
<th>PEN</th>
<th>L2</th>
<th>L1</th>
<th>B2</th>
<th>B1</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>CHARACTER LENGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>5 BITS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PARITY ENABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 = ENABLE 0 = DISABLE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVEN PARITY GEN/CHECK</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 = ODD 1 = EVEN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EXTERNAL SYNC DETECTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 = SYSDET IS AN INPUT</td>
</tr>
<tr>
<td>0 = SYSDET IS AN I/O OUTPUT</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SINGLE CHARACTER SYNC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 = SINGLE SYNC CHARACTER</td>
</tr>
<tr>
<td>0 = DOUBLE SYNC CHARACTER</td>
</tr>
</tbody>
</table>

2. COMMAND INSTRUCTION WORD

This format defines a status word that is used to control the actual operation of 8251. All control words written into 8251 after the mode instruction will load the command instruction.

The command instructions can be written into 8251 at any time in the data block during the operation of the 8251. To return to the mode instruction format, the master reset bit in the command instruction word can be set to initiate an internal reset operation which automatically places the 8251 back into the mode instruction format. Command instructions must follow the mode instructions or sync characters. Thus the control word 37 (HEX) enables the transmit enable and receive enable bits, forces DTR output to zero, resets the error flags, and forces RTS output to zero.
# COMMAND INSTRUCTION FORMAT

<table>
<thead>
<tr>
<th></th>
<th>EH</th>
<th>IR</th>
<th>RTS</th>
<th>ER</th>
<th>SBRK</th>
<th>RXE</th>
<th>DTR</th>
<th>TXEN</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TRANSMIT ENABLE</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = ENABLE 0 = DISABLE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>DATA TERMINAL READY</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HIGH will force DTR Output to Zero</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>RECEIVE ENABLE</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = ENABLE 0 = DISABLE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SEND BREAK CHARACTER</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = Forces TXD LOW 0 = Normal Operation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ERROR RESET</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = Reset Error Flags PE, OE, FE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>REQUEST TO SEND</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HIGH will force RTS Output to Zero</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>INTERNAL RESET</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HIGH Returns 8251 to Mode Instruction Format</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ENTER HUNT MODE</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = Enable a Search for Sync Characters (Has No Effect in Async mode)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ALGORITHM:
1. Initialize timer (8253) IC.
2. Move the mode command word (4E H) to A – reg.
3. Output it to port address C2.
4. Move the command instruction word (37 H) to A – reg.
5. Output it to port address C2.
6. Move the data to be transferred to A – reg.
7. Output it to port address C0.
8. Reset the system.
9. Get the data through input port address C0.
10. Store the value in memory.
11. Reset the system.

PROGRAM:

```
MVI A, 36 H
OUT CE H
MVI A, 0A H
OUT C8 H
MVI A, 00
OUT C8 H
LXI H, 4200
MVI A, 4E
OUT C2
MVI A, 37
OUT C2
MVI A, 41
OUT C0
RST 1

ORG 4200
IN C0
STA 4500
RST 1
```

OBSERVATION:
Output: \((4500) \rightarrow 41^H\)

RESULT:
Thus the 8251 was initiated and the transmission and reception of character was done successfully.
AIM:
To find the sum of elements in an array.

ALGORITHM:

1. Load the array in the consecutive memory location and initialize the memory pointer with the starting address.
2. Load the total number of elements in a separate register as a counter.
3. Clear the accumulator.
4. Load the other register with the value of the memory pointer.
5. Add the register with the accumulator.
6. Check for carry, if exist, increment the carry register by 1. otherwise, continue
7. Decrement the counter and if it reaches 0, stop. Otherwise increment the memory pointer by 1 and go to step 4.
PROGRAM:

    MOV DPTR, #4200
    MOVX A, @DPTR
    MOV R0, A
    MOV B, #00
    MOV R1, B
    INC DPTR

LOOP2:
    CLR C
    MOVX A, @DPTR
    ADD A, B
    MOV B, A
    JNC LOOP
    INC R1

LOOP:
    INC DPTR
    DJNZ R0, LOOP2
    MOV DPTR, #4500
    MOV A, R1
    MOVX @DPTR, A
    INC DPTR
    MOV A, B
    MOVX @DPTR, A

HLT:
    SJMP HLT

INPUT

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4200</td>
<td>4201</td>
</tr>
<tr>
<td>04</td>
<td>05</td>
</tr>
</tbody>
</table>

OUTPUT:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4500</td>
<td>4501</td>
</tr>
<tr>
<td>0F</td>
<td>00</td>
</tr>
</tbody>
</table>

RESULT:

The sum of elements in an array is calculated.
AIM:
To perform hexadecimal to decimal conversion.

ALGORITHM:
1. Load the number to be converted into the accumulator.
2. If the number is less than 100 (64H), go to next step; otherwise, subtract 100 (64H) repeatedly until the remainder is less than 100 (64H). Have the count (100’s value) in separate register which is the carry.
3. If the number is less than 10 (0AH), go to next step; otherwise, subtract 10 (0AH) repeatedly until the remainder is less than 10 (0AH). Have the count (ten’s value) in separate register.
4. The accumulator now has the units.
5. Multiply the ten’s value by 10 and add it with the units.
6. Store the result and carry in the specified memory location.
PROGRAM:

MOV D PTR, #4500
MOVX A, @DPTR
MOV B, #64
DIV A, B
MOV D PTR, #4501
MOVX @DPTR, A
MOV A, B
MOV B, #0A
DIV A, B
INC D PTR
MOVX @DPTR, A
INC D PTR
MOV A, B
MOVX @DPTR, A
HLT: SJMP HLT

INPUT | OUTPUT:
--- | ---
4500 D7 |
4501 15 |
4502 02 |

RESULT:
The given hexadecimal number is converted into decimal number.
**8051 - DECIMAL TO HEXADECIMAL CONVERSION**

**AIM:**
To perform decimal to hexadecimal conversion

**ALGORITHM:**
1. Load the number to be converted in the accumulator.
2. Separate the higher order digit from lower order.
3. Multiply the higher order digit by 10 and add it with the lower order digit.
4. Store the result in the specified memory location.

**PROGRAM:**

```
MOV DPTR, #4500
MOVX A, @DPTR
MOVB, #0A
MUL A, B
MOV B, A
INC DPTR
MOVX A, @DPTR
ADD A, B
INC DPTR
MOVX @DPTR, A
HLT: SJMP HLT
```

**INPUT** | **OUTPUT**
---|---
4500 | 23
4501 | 17

**RESULT:**
The given decimal number is converted to hexadecimal number.
AIM:
To interface a stepper motor with 8051 microcontroller and operate it.

THEORY:
A motor in which the rotor is able to assume only discrete stationary angular position is a stepper motor. The rotary motion occurs in a step-wise manner from one equilibrium position to the next. Stepper Motors are used very wisely in position control systems like printers, disk drives, process control machine tools, etc.

The basic two-phase stepper motor consists of two pairs of stator poles. Each of the four poles has its own winding. The excitation of any one winding generates a North Pole. A South Pole gets induced at the diametrically opposite side. The rotor magnetic system has two end faces. It is a permanent magnet with one face as South Pole and the other as North Pole.

The Stepper Motor windings A1, A2, B1, B2 are cyclically excited with a DC current to run the motor in clockwise direction. By reversing the phase sequence as A1, B2, A2, B1, anticlockwise stepping can be obtained.

2-PHASE SWITCHING SCHEME:
In this scheme, any two adjacent stator windings are energized. The switching scheme is shown in the table given below. This scheme produces more torque.
### ADDRESS DECODING LOGIC:

The 74138 chip is used for generating the address decoding logic to generate the device select pulses, CS1 & CS2 for selecting the IC 74175. The 74175 latches the data bus to the stepper motor driving circuitry.

Stepper Motor requires logic signals of relatively high power. Therefore, the interface circuitry that generates the driving pulses use silicon darlington pair transistors. The inputs for the interface circuit are TTL pulses generated under software control using the Microcontroller Kit. The TTL levels of pulse sequence from the data bus is translated to high voltage output pulses using a buffer 7407 with open collector.

### PROCEDURE:

Enter the above program starting from location 4100 and execute the same. The stepper motor rotates. Varying the count at R4 and R5 can vary the speed. Entering the data in the look-up TABLE in the reverse order can vary direction of rotation.

<table>
<thead>
<tr>
<th>STEP</th>
<th>A1</th>
<th>A2</th>
<th>B1</th>
<th>B2</th>
<th>DATA</th>
<th>STEP</th>
<th>A1</th>
<th>A2</th>
<th>B1</th>
<th>B2</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>9h</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Ah</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>5h</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>6h</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>6h</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>5h</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Ah</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>9h</td>
</tr>
</tbody>
</table>
**PROGRAM:**

<table>
<thead>
<tr>
<th>Address</th>
<th>OPCODES</th>
<th>Label</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>4100</td>
<td>START: MOV</td>
<td>DPTR, #TABLE</td>
<td>Load the start address of switching scheme data TABLE into Data Pointer (DPTR)</td>
</tr>
<tr>
<td>4103</td>
<td>MOV</td>
<td>R0, #04</td>
<td>Load the count in R0</td>
</tr>
<tr>
<td>4105</td>
<td>LOOP: MOVX</td>
<td>A, @DPTR</td>
<td>Load the number in TABLE into A</td>
</tr>
<tr>
<td>4106</td>
<td>PUSH</td>
<td>DPH</td>
<td>Push DPTR value to Stack</td>
</tr>
<tr>
<td>4108</td>
<td>PUSH</td>
<td>DPL</td>
<td></td>
</tr>
<tr>
<td>410A</td>
<td>MOV</td>
<td>DPTR, #0FFC0h</td>
<td>Load the Motor port address into DPTR</td>
</tr>
<tr>
<td>410D</td>
<td>MOVX</td>
<td>@DPTR, A</td>
<td>Send the value in A to stepper Motor port address</td>
</tr>
<tr>
<td>410E</td>
<td>MOV</td>
<td>R4, #0FFh</td>
<td>Delay loop to cause a specific amount of time delay before next data item is sent to the Motor</td>
</tr>
<tr>
<td>4110</td>
<td>DELAY: MOV</td>
<td>R5, #0FFh</td>
<td></td>
</tr>
<tr>
<td>4112</td>
<td>DELAY1: DJNZ</td>
<td>R5, DELAY1</td>
<td></td>
</tr>
<tr>
<td>4114</td>
<td>DJNZ</td>
<td>R4, DELAY</td>
<td></td>
</tr>
<tr>
<td>4116</td>
<td>POP</td>
<td>DPL</td>
<td>POP back DPTR value from Stack</td>
</tr>
<tr>
<td>4118</td>
<td>POP</td>
<td>DPH</td>
<td></td>
</tr>
<tr>
<td>411A</td>
<td>INC</td>
<td>DPTR</td>
<td>Increment DPTR to point to next item in the table</td>
</tr>
<tr>
<td>411B</td>
<td>DJNZ</td>
<td>R0, LOOP</td>
<td>Decrement R0, if not zero repeat the loop</td>
</tr>
<tr>
<td>411D</td>
<td>SJMP</td>
<td>START</td>
<td>Short jump to Start of the program to make the motor rotate continuously</td>
</tr>
<tr>
<td>411F</td>
<td>TABLE: DB</td>
<td>09 05 06 0Ah</td>
<td>Values as per two-phase switching scheme</td>
</tr>
</tbody>
</table>

**RESULT:**

Thus a stepper motor was interfaced with 8051 and run in forward and reverse directions at various speeds.