MUFFAKHAM JAH
COLLEGE OF ENGINEERING AND TECHNOLOGY

PC455EC MICROPROCESSOR & MICROCONTROLLER LAB MANUAL

(With effect from the academic year 2022-2023)

**DEPARTMENT OF
ELECTRONICS AND COMMUNICATION ENGINEERING**

Vision and Mission of the Institution

Vision

To be part of universal human quest for development and progress by contributing high calibre, ethical and socially responsible engineers who meet the global challenge of building modern society in harmony with nature.

Mission

- To attain excellence in imparting technical education from the undergraduate through doctorate levels by adopting coherent and judiciously coordinated curricular and co-curricular programs
- To foster partnership with industry and government agencies through collaborative research and consultancy
- To nurture and strengthen auxiliary soft skills for overall development and improved employability in a multi-cultural work space
- To develop scientific temper and spirit of enquiry in order to harness the latent innovative talents
- To develop constructive attitude in students towards the task of nation building and empower them to become future leaders
- To nourish the entrepreneurial instincts of the students and hone their business acumen.
- To involve the students and the faculty in solving local community problems through economical and sustainable solutions.

Vision and Mission of ECE Department

Vision

To be recognized as a premier education center providing state of art education and facilitating research and innovation in the field of Electronics and Communication.

Mission

We are dedicated to providing high quality, holistic education in Electronics and Communication Engineering that prepares the students for successful pursuit of higher education and challenging careers in research, R& D and Academics.

Program Educational Objectives of B. E (ECE) Program:

1. Graduates will demonstrate technical competence in their chosen fields of employment by identifying, formulating, analyzing and providing engineering solutions using current techniques and tools
2. Graduates will communicate effectively as individuals or team members and demonstrate leadership skills to be successful in the local and global cross-cultural working environment
3. Graduates will demonstrate lifelong learning through continuing education and professional development
4. Graduates will be successful in providing viable and sustainable solutions within societal, professional, environmental and ethical contexts

MUFFAKHAM JAH
COLLEGE OF ENGINEERING AND TECHNOLOGY
BANJARA HILLS, ROAD NO-3, TELANGANA-500034

LABORATORY MANUAL
FOR
MICROPROCESSOR & MICROCONTROLLER LAB

Prepared by: HAKEEM AEJAZ ASLAM **Checked by: J.K.NAG**

Approved by: Dr. ARIFUDDIN SOHEL

MUFFAKHAM JAH
COLLEGE OF ENGINEERING AND TECHNOLOGY
DEPARTMENT OF ELECTRONICS AND COMMUNICATIONS ENGINEERING
(Name of the Subject/Lab Course): Microprocessor & Microcontroller Lab

Code: PC453EC	**Programme: UG**
Branch: ECE	**Version No: 1**
Year : III	**Updated on: 03/8/22**
Semester: V	**No. of Pages: 112**

Classification Status (Unrestricted/restricted): Unrestricted
Distribution List: Department, Lab, Library, Lab Incharge

Prepared by: 1) Name:
2) Sign :
3) Designation:
4) Date :

1) Name:
2) Sign :
3) Designation:
4) Date :

Verified by: 1) Name:
2) Sign :
3) Designation:
4) Date :

*** For Q.C Only**
1) Name:
2) Sign :
3) Designation:
4) Date :

Approved by: (HOD) 1) Name:
2) Sign :
3) Date :

List of Experiments

PART- A

1. Use of 8086 trainer kit and execution of programs. (Instruction set for simple Programs using 4 to 5 lines of instruction code under different addressing modes for data transfer, manipulation, Arithmetic operations)
2. Branching operations and logical operations in a given data.
   i) transfer byte and word data from source to destination memory.
   ii) Count even and odd numbers from given Array of ten bytes.
   iii) Find Largest and Smallest number from given array of words
   iv) Sort the given array in ascending order, descending order
3. Multiplication and Division
   i) Use MUL and IMUL for Unsigned and signed multiplication on 8 bit and 16 bit data sets
   ii) Use DIV and IDIV for Unsigned and signed division on 8 bit and 16bit data sets
   iii) Obtain given decimal number to unpacked BCD ex: 1234_{10} as 01,02,03,04 and store in memory using DIV
   iv) Find Factorial of a given number using multiplication instructions
4. Single byte, multi-byte Binary and BCD addition and subtraction
5. Code conversions.
   i) BCD Unpacked to Packed BCD
   ii) Ascii code to BCD code
   iii) BCD to Ascii
6. String Searching and Sorting.(Using string instructions)
   i) Find number of repetitions of a character in a string
   ii) Find and replace a character in the given string
   iii) Convert Case of a given string
   iv) Find whether given string is palindrome or not

Part B

[Experiments for 8051 using any C- Cross Compiler & appropriate hardware]

1. Familiarity and use of 8051/8031 Microcontroller trainer, and execution of programs.
2. Instruction set for simple Programs (using 4 to 5 lines of instruction code).
3. Timer and counter operations & programming using 8051.
4. Serial communications using UART
5. Programming using interrupts
6. Interfacing 8051 with DAC to generate waveforms.

7. Interfacing traffic signal control using 8051.
8. Program to control stepper motor using 8051.
9. ADC interfacing with 8051
10. Serial RTC interfacing with 8051
11. LCD interfacing with 8051

NOTE: 1. At least ten experiments to be conducted in the semester.
2. Minimum of 5 from Part A and 5 from Part B is compulsory.
3. In Part-B, perform the experiments using assembler simulators like edsim51/Keil Software.

MICROPROCESSOR & MICROCONTROLLER LAB
GENERAL GUIDELINES AND SAFETY INSTRUCTIONS

1. Sign in the log register as soon as you enter the lab and strictly observe your lab timings.
2. Strictly follow the written and verbal instructions given by the teacher / Lab Instructor. If you do not understand the instructions, the handouts and the procedures, ask the instructor or teacher.
3. **Never work alone!** You should be accompanied by your laboratory partner and / or the instructors / teaching assistants all the time.
4. It is mandatory to come to lab in a formal dress and wear your ID cards.
5. Do not wear loose-fitting clothing or jewelry in the lab. Rings and necklaces are usual excellent conductors of electricity.
6. Mobile phones should be switched off in the lab. Keep bags in the bag rack.
7. Keep the labs clean at all times, no food and drinks allowed inside the lab.
8. Intentional misconduct will lead to expulsion from the lab.
9. Do not handle any equipment without reading the safety instructions. Read the handout and procedures in the Lab Manual before starting the experiments.
10. Do your wiring, setup, and a careful circuit checkout before applying power. Do not make circuit changes or perform any wiring when power is on.
11. Avoid contact with energized electrical circuits.
12. Do not insert connectors forcefully into the sockets.
13. **NEVER** try to experiment with the power from the wall plug.
14. Immediately report dangerous or exceptional conditions to the Lab instructor/ teacher: Equipment that is not working as expected, wires or connectors are broken, the equipment that smells or “smokes”. If you are not sure what the problem is or what's going on, switch off the Emergency shutdown.
15. Never use damaged instruments, wires or connectors. Hand over these parts to the Lab instructor/Teacher.
16. Be sure of location of fire extinguishers and first aid kits in the laboratory.
17. After completion of Experiment, return the bread board, trainer kits, wires, CRO probes and other components to lab staff. Do not take any item from the lab without permission.
18. Observation book and lab record should be carried to each lab. Readings of current lab experiment are to be entered in Observation book and previous lab experiment should be written in Lab record book. Both the books should be corrected by the faculty in each lab.
19. Handling of Semiconductor Components: Sensitive electronic circuits and electronic components have to be handled with great care. The inappropriate handling of electronic component can damage or destroy the devices. The devices can be destroyed by driving to high currents through the device, by overheating the device, by mixing up the polarity, or by electrostatic discharge (ESD). Therefore, always handle the electronic devices as indicated by the handout, the specifications in the data sheet or other documentation.
20. Special Precautions during soldering practice

    a. Hold the soldering iron away from your body. Don't point the iron towards you.

    b. Don't use a spread solder on the board as it may cause short circuit.

    c.Do not overheat the components as excess heat may damage the components/board.

d. In case of burn or injury seek first aid available in the lab or at the college dispensary.

MUFFAKHAM JAH COLLEGE OF ENGINEERING & TECHNOLOGY
BE ¾ Year Vth Semester ECE
Microprocessor and Microcontroller Lab

List of Experiments

PART A

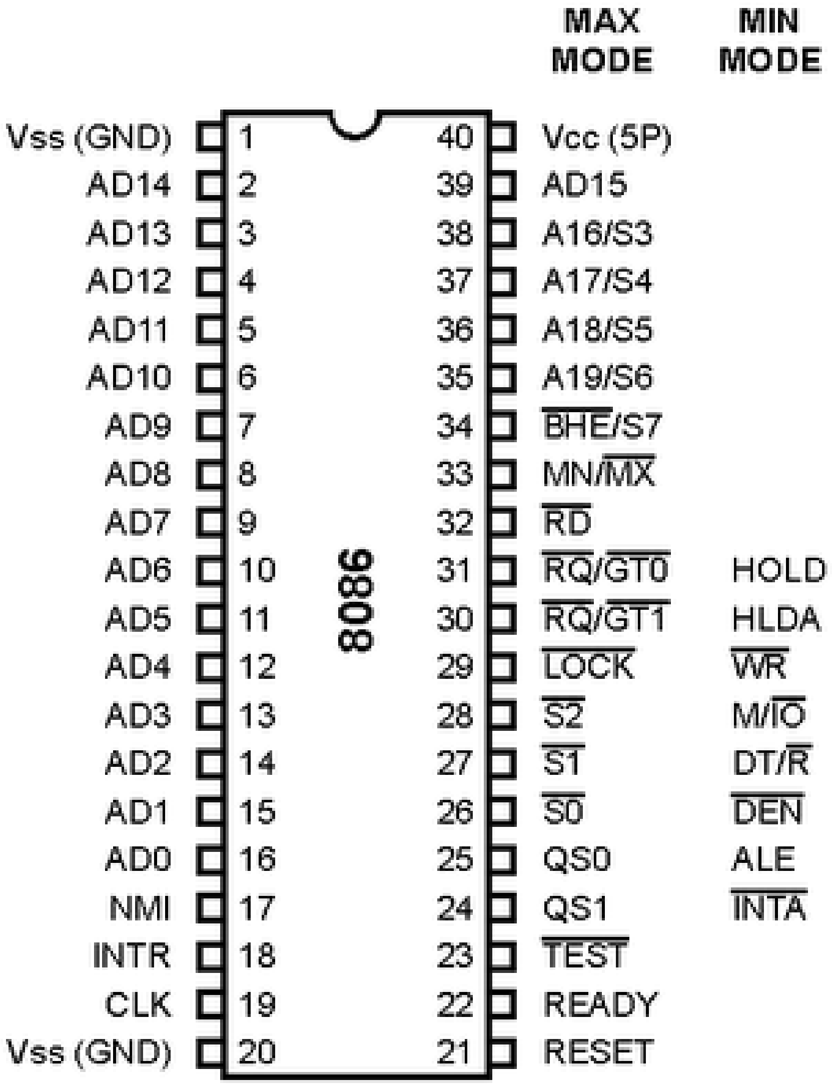
[Experiments on assembly language programming for 8086 using Assembler]

PART B

[Experiments on assembly language programming for 8051 using Assembler]

PART A

[Experiments on assembly language programming for 8086 using Assembler]



8086 Microprocessor pin diagram

8086 Instruction Set

Data Transfer Instructions:

MOV	Move byte or word to register or memory
IN, OUT	Input byte or word from port, output word to port
LEA	Load effective address
LDS, LES	Load pointer using data segment, extra segment
PUSH, POP	Push word onto stack, pop word off stack
XCHG	Exchange byte or word
XLAT	Translate byte using look-up table

Logical Instructions:

NOT	Logical NOT of byte or word (one's complement)
AND	Logical AND of byte or word
OR	Logical OR of byte or word
XOR	Logical exclusive-OR of byte or word
TEST	Test byte or word (AND without storing)

Shift and Rotate Instructions:

SHL, SHR	Logical shift left, right byte or word by 1 or CL
SAL, SAR	Arithmetic shift left, right byte or word by 1 or CL
ROL, ROR	Rotate left, right byte or word by 1 or CL
RCL, RCR	Rotate left, right through carry byte or word by 1 or CL

Arithmetic Instructions:

ADD, SUB	Add, subtract byte or word
ADC, SBB	Add, subtract byte or word and carry (borrow)
INC, DEC	Increment, decrement byte or word
NEG	Negate byte or word (two's complement)
CMP	Compare byte or word (subtract without storing)
MUL, DIV	Multiply, divide byte or word (unsigned)
IMUL, IDIV	Integer multiply, divide byte or word (signed)
CBW, CWD	Convert byte to word, word to double word (useful before multiply/divide)

Adjustments after arithmetic operations:

AAA, AAS, AAM, AAD	ASCII adjust for addition, subtraction, multiplication, division (ASCII codes 30-39)
DAA, DAS	Decimal adjust for addition, subtraction (Binary coded decimal numbers)

Transfer Instructions:

JMP Unconditional jump (short 127/8, near 32K, far between segments)

Conditional jumps:

JA (JNBE)	Jump if above (not below or equal) +127, -128 range only
JAE (JNB)	Jump if above or equal (not below) +127, -128 range only
JB (JNAE)	Jump if below (not above or equal) +127, -128 range only
JBE (JNA)	Jump if below or equal (not above) +127, -128 range only
JE (JZ)	Jump if equal (zero) +127, -128 range only

JG (JNLE)	Jump if greater (not less or equal) +127, -128 range only
JGE (JNL)	Jump if greater or equal (not less) +127, -128 range only
JL (JNGE)	Jump if less (not greater nor equal) +127, -128 range only
JLE (JNG)	Jump if less or equal (not greater) +127, -128 range only
JC, JNC	Jump if carry set, carry not set +127, -128 range only
JO, JNO	Jump if overflow, no overflow +127, -128 range only
JS, JNS	Jump if sign, no sign +127, -128 range only
JNP (JPO)	Jump if no parity (parity odd) +127, -128 range only
JP (JPE)	Jump if parity (parity even) +127, -128 range only

Loop control:

LOOP	Loop unconditional, count in CX, short jump to target address
LOOPE (LOOPZ)	Loop if equal (zero), count in CX, short jump to target address
LOOPNE (LOOPNZ)	Loop if not equal (not zero), count in CX, short jump to target address
JCXZ	Jump if CX equals zero (used to skip code in loop)

Subroutine and Interrupt Instructions:

CALL, RET	Call, return from procedure (inside or outside current segment)
INT, INTO	Software interrupt, interrupt if overflow
IRET	Return from interrupt

String Instructions:

MOVS	Move byte or word string
MOVSB, MOVSW	Move byte, word string
CMPS	Compare byte or word string
SCAS	Scan byte or word string (comparing to A or AX)
LODS, STOS	Load, store byte or word string to AL or AX

Repeat instructions (placed in front of other string operations):

REP	Repeat
REPE, REPZ	Repeat while equal, zero
REPNE, REPNZ	Repeat while not equal (zero)

Processor Control Instructions:

Flag manipulation:

STC, CLC, CMC	Set, clear, complement carry flag
STD, CLD	Set, clear direction flag
STI, CLI	Set, clear interrupt enable flag
LAHF, SAHF	Load AH from flags, store AH into flags
PUSHF, POPF	Push flags onto stack, pop flags off stack

Coprocessor, multiprocessor interface:

ESC	Escape to external processor interface
LOCK	Lock bus during next instruction

Inactive states:

NOP	No operation
WAIT	Wait for TEST pin activity
HLT	Halt processor

Experiment No: 1

Study & use of 8086 trainer kit and execution of programs

Central Processors:-

8086 or 8088 CPU operating at 5 MHz in maximum mode (supplied with 8086 CPU).

Co-Processor:-

Onboard 8087 Numeric Data Processor (optional).

Memory:-

ESA 86/88E provides a total of 128K Bytes of onboard memory.

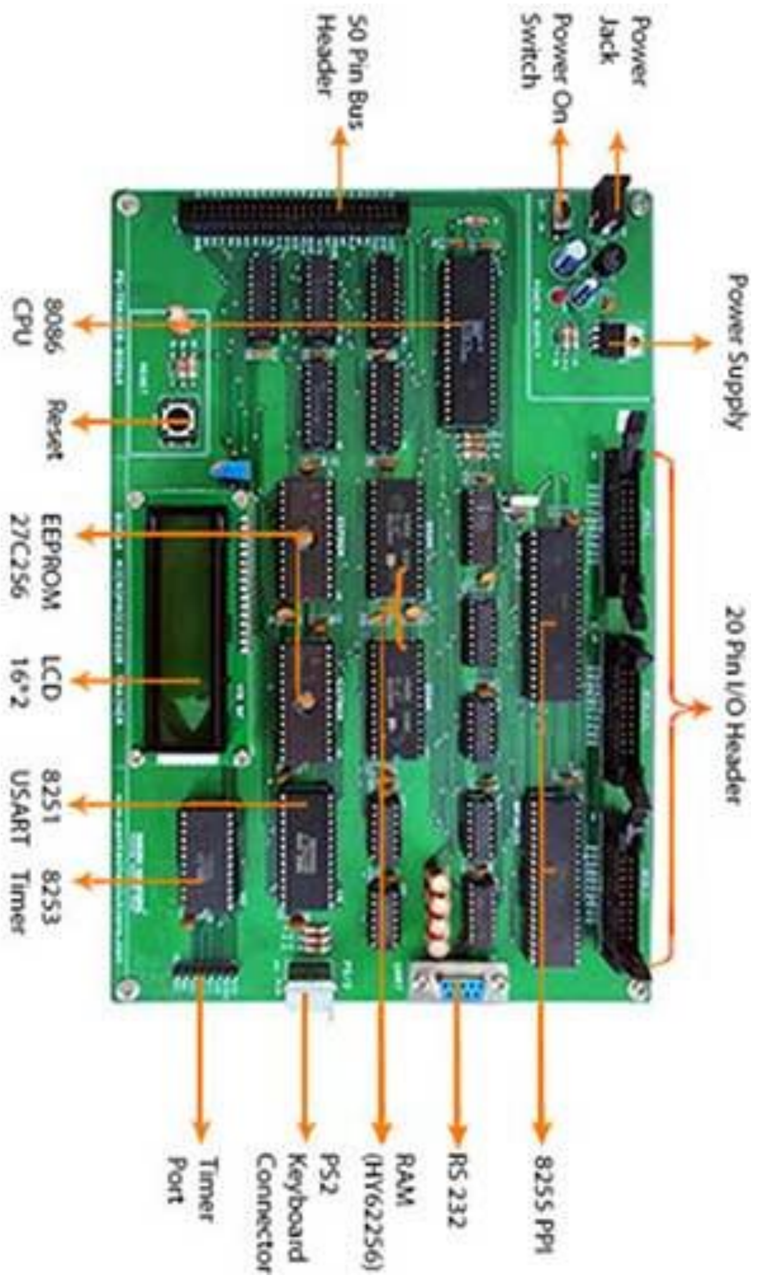
64K Bytes of ROM using two 27256 EPROMs
64K Bytes of RAM using two 62256 static RAMs

Memory Addressing:-

ESA 86/88E memory is arranged in odd and even banks of memory.
ESA 86/88E has four 28 pin JEDEC compatible slots labeled as U4-U7 for memory IC's.

Sockets U6 & U7 are populated with EPROM'S 27256 (32k*2=64K Bytes) or 27512 (64K*2=128K Bytes) containing system firmware.

Sockets U4& U5 re populated with SRAM'S 62256 (32k*2=64K Bytes). RAM area starts from 0000H. However the RAM from 0000H to 1FFFH is used by the system for interrupt vectors, stack and Assembler data tables. Thus user RAM area starts from location 2000H onwards.



Microprocessor 8086 Trainer kit

Memory Map:-

Memory Type	Sockets Used	Device	Address Range
EPROM	U6 & U7	27256	F0000-FFFFF
RAM	U4 & U5	62256	00000-0FFFF

Optional battery backup provision is available for RAM using onboard 3.6V Ni-Cd cell.

System Timing:-

The time base for CPU operation is derived from an 8284A clock generator. The CPU operates at a frequency of 5MHz, which is the output from the clock generator with 33% duty cycle.

The clock generator also generates output signal at 2.5MHz pulse Clock with 50% duty cycle.

CPU RESET:-

A 15MHz crystal is a clock source for the 8284A Clock generator. The 8284A divides the frequency by thrice and produces a 5MHz clock with 33% duty cycle as required by 8086/8088. Further, 8284A provides a 2.5MHz pulse clock with 50% duty cycle which can be used as a clock input for onboard peripherals.

Both these clock outputs are available on the bus connector and may be used as the source clock frequency to external peripherals.

CPU Address Bus:-

Latches (74LS373's) at U15, U16, U17 are used to latch the address with the help of ALE signal. As the CPU operates in the maximum mode, the 8288 bus controller is used to decode the state signals and provide the entire control signal.

CPU Data Bus:-

Bi-Directional buffers (74LS245's) at U14, U21 & U27 used to buffer the CPU data bus.

Interrupt system:-

Hardware interrupts: -

External:-

The 8086/88 CPU supports two Hardware interrupts NMI and INTR.

NMI: 8086/88 Type 3 interrupt connected to KBINT on the trainer. The vectoring information for this interrupt is fully user defined.

INTR: This line is left unconnected.

Internal:-

INT 3 can be used by user programs to return control to monitor.

Interrupt vector 1 (Single step interrupt) and 3 (Break point interrupt) reserved for monitor.

Other internal interrupts are available to the user.

Onboard Peripherals & Interfacing options:-

8251A:- USART supporting standard baud rates from 110 to 19200. Baud rate is selected through onboard DIP (Dual In-Line) switch settings.
8253:- Programmable interval Timer, Timer 0 is used for Baud clock generation. Timer 1 and Timer 2 are available to the user.
8255A:- 3 PPI, provides up to 72 programmable I/O lines. One 8255 is used for controlling LCD and reading DIP switch. Two 8255's are for the user of which one is populated by default and the other is optional.
8288:- Bus controller used for generating control signals in maximum mode operation.
8042/8742:- UPI (Universal Peripheral Interface) or 8274 adapter for PC Keyboard interfacing.

External Interfacing Signals:-

CPU Bus:- De-multiplexed and fully buffered TTL compatible, address, data and control signals available on a 50 pin ribbon cable connector.

Parallel I/O:- 48 programmable parallel I/O lines (TTL compatible) through two 26 pin ribbon cable connector.

Serial I/O:- RS232 through onboard 9 pin D-type female connector.
PC keyboard DIN connector is provided for interfacing PC keyboard.

20x4 LCD:- 15 pin flow strip for interfacing the LCD.

Timer Signals:- Timer 1 & Timer 2 signals are brought to a header.

Power Supply:- +5V@950mA(approximately)

Battery Back-up:- 3.6V Ni-Cd battery for power back up to RAM (Optional).

Commands used on ESA 86/88E

- **Assemble command**

  A 2000

- **To come out of assembler command**

  !

- **To run program**

  G 2000

- **Examining/modifying memory location**

  S result location/memory location

- **Disassembly command**

  Z 2000, ending address

- **Stand Alone mode or Keyboard mode DIP 7 → ON**
- **Serial Communication mode DIP 3, 4 & 7 → ON**

Simple Programs

Program 1: Write a program to add two 16-bit numbers using registers and place the result in other register ignoring the possible overflow.

Algorithm:

1. Initialize two 16-bit data in registers
2. Add the data in registers
3. Store the result in some other register
4. End the program

Source Code:

```
Output 2500ad      ; Linker version command
Org 2000h          ; Starting address
Mov AX,1234h       ; Load first 16-bit data in reg
Mov DX,8765h       ; Load second 16-bit data in other reg
Add AX,DX          ; Add two 16-bit regs
Mov CX,AX          ; Store the result
Int 3              ; Halt
```

Result:

```
Input:   AX = 1234h                Output:   AX = 9999h
         DX = 8765h                          CX = 9999h
```

Program 2: Write a program to subtract two 16-bit numbers using registers and place the result in other register.

Algorithm:

1. Initialize two 16-bit data in registers
2. Subtract the data in registers
3. Store the result in some other register
4. End the program

Source Code:

```
Output 2500ad          ; Linker version command
   Org 2000h              ; Starting address
Mov AX,9876h           ; Load first 16-bit data in reg
Mov DX,1234h           ; Load second 16-bit data in other reg
Sub AX,DX              ; Subtract two 16-bit regs
Mov CX,AX              ; Store the result
Int 3                  ; Halt
```

Result:

```
Input:   AX = 9876h                    Output:   AX = 8642h
         DX = 1234h                              CX = 8642h
```

Program 3(a): Write a program to multiply two 8-bit numbers using registers and place the result in other registers.

Algorithm:

1. Initialize two 8-bit data in registers
2. Multiply the data in registers
3. Store the result in some other register
4. End the program

Source Code:

```
Output 2500ad        ; Linker version command
Org 2000h            ; Starting address
Mov AL,03h           ; Load 8-bit multiplicand in reg
Mov CL,05h           ; Load 8-bit multiplier in other reg
Mul CL               ; Multiply two 8-bit regs
Mov DL,AL            ; Store the result
Int 3                ; Halt
```

Result:

```
Input:   AL = 03h                Output:   AL = 0Fh
         CL = 05h                          DL = 0Fh
```

Program 3(b): Write a program to multiply two 16-bit numbers using registers and place the result in other registers.

Algorithm:
1. Initialize two 16-bit data in registers
2. Multiply the data in registers
3. Store the result in some other register
4. End the program

Source Code:

```
Output 2500ad          ; Linker version command
Org 2000h              ; Starting address
Mov AX,006fh           ; Load 16-bit multiplicand in reg
  Mov DX,0061h           ; Load 16-bit multiplier in other reg
Mul DX                 ; Multiply two 16-bit regs
Mov CX,AX              ; Store the result
Int 3                  ; Halt
```

Result:

```
Input:   AX = 006Fh            Output:   AX = 2A0Fh
         DX = 0061h                      CX = 2A0Fh
```

Program 4(a): Write a program to divide two 8-bit numbers using registers and place the result in other registers.

Algorithm:

1. Initialize two 8-bit data in registers
2. Divide the data in registers
3. Store the result in some other register
4. End the program

Source Code:

```
Output 2500ad        ; Linker version command
Org 2000h            ; Starting address
Mov AH,00h           ; Clear AH reg
Mov AL,43h           ; Load 8-bit dividend in reg
Mov CL,08h           ; Load 8-bit divisor in other reg
Div CL               ; Divide two 8-bit regs
Int 3                ; Halt
```

Result:

```
Input:   AL = 43h              Output:   AX = 0308h
         CL = 08h
```

Q = AL = 08h

R = AH = 03h

Program 4(b): Write a program to divide two 16-bit numbers using registers and place the result in other registers.

Algorithm:

1. Initialize two 16-bit data in registers
2. Divide the data in registers
3. Store the result in some other register
4. End the program

Source Code:

```
Output 2500ad          ; Linker version command
Org 2000h              ; Starting address
Mov DX,0000h           ; Clear DX reg
Mov AX,0045h           ; Load 16-bit dividend in reg
Mov CX,0008h           ; Load 16-bit divisor in other reg
Div CX                 ; Divide two 8-bit regs
Int 3                  ; Halt
```

Result:

```
Input:   AX = 0045h                Output:   AX = 0008h
         CX = 0008h                          DX = 0005h
```

DX = 0005h & AX = 0008h

Q = AX = 0008h

R = DX = 0005h

Program 5: Write a program to add two 8-bit decimal numbers using registers.

Algorithm:

1. Initialize two 8-bit data in registers
2. Add the data in registers
3. Convert hex data into decimal data
4. Store the result in some other register
5. End the program

Source Code:

```
Output 2500ad          ; Linker version command
Org 2000h              ; Starting address
Mov AL,07h             ; Load first 8-bit data in reg
   Mov DL,15h             ; Load second 8-bit data in other reg
Add AL,DL              ; Add two 8-bit regs
Daa                    ; Decimal adjust after addition
Mov CL,AL              ; Store the result
Int 3                  ; Halt
```

Result:

```
Input:   AL = 07h                  Output:   AL = 22D
         DL = 15h                            CL = 22D
```

Program 6: Write a program to subtract two 8-bit decimal numbers using registers.

Algorithm:

1. Initialize two 8-bit data in registers
2. Subtract the data in registers
3. Convert hex data into decimal data
4. Store the result in some other register
5. End the program

Source Code:

```
Output 2500ad          ; Linker version command
Org 2000h              ; Starting address
Mov AL,32h             ; Load first 8-bit data in reg
Mov DL,19h             ; Load second 8-bit data in other reg
Sub AL,DL              ; Subtract two 8-bit regs
Das                    ; Decimal adjust after subtraction
Mov CL,AL              ; Store the result
Int 3                  ; Halt
```

Result:

```
Input:   AL = 32h                 Output:   AL = 13D
         DL = 19h                           CL = 13D
```

EXPERIMENT- 2

Programs using different addressing modes

Addressing modes defines the way of representing the data in registers or memory locations. The different addressing modes in 8086 microprocessor are as follows:

1. IMMEDIATE ADDRESSING MODE:

In this type of addressing immediate data is a part of instruction, and appears in the form of successive byte or bytes.

EX: Mov AX,1234h

```
Mov BX,ABCDh
Mov CL, 08h
Mov DH, 12h
```

In the above example, 1234h is the immediate data. The immediate data may be 8-bit or 16-bit in size.

2. DIRECT ADDRESSING MODE:

In the direct addressing mode, a 16-bit memory address (offset) is directly specified in the instruction as a part of it.

EX: Mov AL, [3000h]

```
Mov AH, [3100h]
```

In the above example, the data stored in the memory location 3000h is moved into AX register that is, the contents of memory location 3000h is stored in AL and the contents of memory location 3001h is stored in AH.

3. REGISTER ADDRESSING MODE:

In register addressing mode, the data is stored in a register and it is referred using the particular register. All the registers, except IP (instruction pointer) may be used.

EX: Mov BX, AX

In the above example, a 16-bit data which is there in AX register is moved into BX register. Both the source and destination are registers only.

4. REGISTER INDIRECT ADDRESSING MODE:

Sometimes, the address of the memory location which contains data or operand is determined in an indirect way using the offset registers. This mode of addressing is known as register indirect addressing mode. In this addressing mode, the offset address of the data is in either BX or SI or DI registers. The data is supposed to be available at the address pointed to by the content of any of the above registers.

EX: Mov AL, [BX]
Mov AL, [SI]
Mov AL, [DI]

In the above example, the data stored in the memory location pointed by BX register is moved into AX register.

5. INDEXED ADDRESSING MODE:

In this addressing mode, offset of the operand is stored in one of the indexed registers that is, SI or DI. This mode is a special case of the above discussed register indirect addressing mode.

EX: Mov AL, [SI]
Mov BL, [DI]

Here the data is available in an offset address stored in SI or DI.

6. Register Relative:

In this addressing mode, the data is available at an effective address formed by adding an 8-bit or 16-bit displacement with the content of any one of the registers BX, BP, SI and DI in the default (either DS or ES) segment. The example given before explains this mode.

Ex: MOV AX, 50h [BX]

Here, effective address is given as 10h*DS + 50h + [BX].

7. Based Indexed: The effective address of data is formed, in this addressing mode, by adding content of a base register (any one of BX or BP) to the content of an index register (any one of SI or DI). The default segment register may be ES or DS.

Ex: MOV AX, [BX] [SI]

Here, BX is the base register and SI is the index register. The effective address is computed as 10h*DS + [BX] + [SI].

8. Relative Based Indexed: The effective address is formed by adding an 8-bit or 16-bit displacement with the sum of contents of any one of the base registers (BX or BP) and any one of the index registers, in a default segment.

Ex: MOV AX, 50h [BX] [SI]

Here, 50h is an immediate displacement, BX is a base register and SI is an index register. The effective address of data is computed as

10h*DS+ [BX] + [SI] + 50h

PROGRAM-7: Write an 8086 program to copy a 16-bit value into the register or memory location using different addressing modes.

Algorithm:

1. Immediate data is moved in some register (Immediate addressing mode)
2. Initialize the data in a register & move it to some other register (Register addressing mode)
3. Initialize the address by using any pointer register, place the data in Acc register & move it from register to the pointer address indirectly (Register indirect addressing mode)
4. Initialize the address by using only index pointer register, place the data in Acc register & move it from register to the pointer address (Indexed addressing mode)
5. End the program

Source Code:

```
Immediate:            Output 2500ad          ; Linker version command
                     Org 2000h              ; Starting address
                     Mov AX,1234h           ; Load immediate data in AX reg
                     Int 3                  ; Halt

Register:             Output 2500ad          ; Linker version command
                     Org 2000h              ; Starting address
                     Mov AX,9876h           ; Load immediate data in AX reg
                     Mov BX,AX              ; Move data in other reg
                     Int 3                  ; Halt


Register indirect:    Output 2500ad          ; Linker version command
                     Org 2000h              ; Starting address
                     Mov BX,4000h           ; Initialize the pointer BX
                     Mov AX, 6789h          ; Load immediate data in AX reg
                      Mov [BX], AL          ; Move AL value to location pointed by BX
                     Mov DL, [BX]           ; Move the value indirectly to DL reg
                     Int 3                  ; Halt
```

Indexed:

```
  Output 2500ad           ; Linker version command
Org 2000h                ; Starting address
Mov SI,4000h             ; Initialize the pointer reg SI
Mov AX, 4321h            ; Load immediate data in AX reg
Mov [SI], AL              ; Move AL value to location pointed by SI
Mov DL, [SI]              ; Move the value indirectly to DL reg
Int 3                    ; Halt
```

PROGRAM-8: Write an 8086 program to copy 35h into memory locations 4000h to 4004h using register indirect addressing mode using:

a) Without a loop and b) With a loop

Algorithm:

1. Initialize the pointer with a memory location
2. Initialize the register with data which has to be moved
3. Move the data from register to the pointer location
4. Increment the pointer and move the data to the pointer location for 'n' times (without a loop) & initialize the counter and repeat the loop for 'n' times (with a loop)
5. End the program

Source Code:

a) Without a loop:

```
Output 2500ad          ; Linker version command
Org 2000h              ; Starting address
Mov SI,4000h           ; Initialize the pointer
Mov AL,35h             ; Load the data in AL reg
Mov [SI], AL           ; Move the data from AL to pointer location
Inc SI                 ; Increment the pointer
Mov [SI], AL           ; Repeat the process 'n' times
Inc SI
Mov [SI], AL
Inc SI
Mov [SI], AL
Inc SI
Mov [SI], AL
Int 3                  ; Halt
```

Result:

```
Input: AL – 35h                    Output: 4000 – 35h
                                           4001 – 35h
                                           4002 – 35h
                                           4003 – 35h
                                           4004 – 35h
```

b) With a loop:

```
              Output 2500ad          ; Linker version command
              Org 2000h              ; Starting address
              Mov SI, 4000h          ; Initialize the pointer
              Mov AL, 35h            ; Load the data in AL reg
              Mov CL,05h             ; Initialize the count
       BACK:  Mov [SI], AL           ; Move the data from AL to pointer location
              Inc SI                 ; Increment the pointer
              Loop BACK              ; Repeat the process ‘n’ times
              Int 3                  ; Halt
```

Result:

```
Input: AL – 35h                    Output: 4000 – 35h
                                           4001 – 35h
                                           4002 – 35h
                                           4003 – 35h
                                           4004 – 35h
```

PROGRAM-9: Write a program to move a source data block starting at address location 3000h to a destination block whose address is 4000h. The length of the source block is in CX register.

Algorithm:
1.Initialize the source pointer, destination pointer & a counter register
2.Move the data from source pointer register to the Acc register
3.Move the data from Acc register to the destination pointer location
4.Increment both the pointer registers
5.Repeat the loop for ‘n’ times
6.End the program

Source Code:

```
        Output 2500ad        ; Linker version command
        Org 2000h            ; Starting address
        Mov SI,3000h         ; Initialize the source pointer
        Mov DI,4000h         ; Initialize the destination pointer
        Mov CX,0008h         ; Initialize the count
   BACK:Mov AL,[SI]          ; Move data from source location to AL reg
        Mov [DI],AL          ; Move the data from AL to dstn pointer
        Inc SI               ; Increment the source pointer
        Inc DI               ; Increment the destination pointer
        Loop BACK            ; Repeat the process ‘n’ times
        Int 3                ; Halt
```

Result:

Input:	3000	01h	Output:	4000	01h
	3001	02h		4001	02h
	3002	03h		4002	03h
	3003	04h		4003	04h
	3004	05h		4004	05h
	3005	06h		4005	06h
	3006	07h		4006	07h
	3007	08h		4007	08h

EXPERIMENT- 3
Multiplication and Division using memory locations

PROGRAM- 10: Write an 8086 program to multiply two 8-bit numbers i.e., multiplicand and multiplier which are present in the memory locations 3000h and 3001h respectively. Store the result in the memory locations 4000h and 4001h.

Algorithm:
1. Initialize the source pointer & destination pointer with a memory location
2. Move the multiplicand stored in source pointer location in lower byte of Acc register
3. Increment the source pointer memory location
4. Move the multiplier stored in source pointer location in lower byte of other register
5. Multiply the data of both registers
6. Store the lower byte of the result in destination location, increment the destination pointer & store the upper byte of the result in destination location
7. End the program

Source Code:

```
Output 2500ad       ; Linker version command
Org 2000h           ; Starting address
Mov SI, 3000h       ; Initialize the source pointer
Mov DI, 4000h       ; Initialize the destination pointer
Mov AX, 0000h       ; Clear AX reg
Mov BX, 0000h       ; Clear BX reg
Mov AL, [SI]        ; Move multiplicand to AL reg
Inc SI              ; Increment source pointer
Mov BL, [SI]        ; Move multiplier to BL reg
Mul BL              ; Multiply
Mov [DI], AL        ; Store lower byte result in dstn loc
Inc DI              ; Increment destination pointer
Mov [DI], AH        ; Store higher byte result in dstn loc
Int 3               ; Halt
```

Result:

```
Input: 3000   12h        Output:  4000   A8h
       3001   34h                 4001   03h
```

PROGRAM- 11: Write an 8086 program to multiply two 16-bit numbers i.e. multiplicand in 3000h and 3001h and multiplier in 3002h and 3003h respectively. Store the result in memory locations 4000h to 4003h.

Algorithm:
1.Initialize the source pointer & destination pointer with a memory location
2.Move the multiplicand stored in source pointer location in Acc register
3.Increment the source pointer memory location
4.Move the multiplier stored in source pointer location in other register
5.Multiply the data of both registers
6.Store the lower word of the result in destination location, increment the destination pointer & store the upper word of the result in destination location
7.End the program

Source Code:

```
Output 2500ad           ; Linker version command
Org 2000h               ; Starting address
Mov SI, 3000h           ; Initialize the source pointer
Mov DI, 4000h           ; Initialize the destination pointer
Mov AX, 0000h           ; Clear AX reg
Mov BX, 0000h           ; Clear BX reg
Mov DX,0000h            ; Clear DX reg
Mov AL, [SI]            ; Move multiplicand to AX reg
Inc SI
Mov AH, [SI]
Inc SI                  ; Increment source pointer
Mov BL, [SI]            ; Move multiplier to BX reg
Inc SI
Mov BH, [SI]
Mul BX                  ; Multiply
Mov [DI], AL            ; Store lower word in dstn loc
Inc DI
Mov [DI], AH
```

```
Inc DI                  ; Increment destination pointer
Mov [DI], DL            ; Store higher word in dstn loc
Inc DI
Mov [DI], DH
Int 3                   ; Halt
```

Result:

```
Input:   3000      34h        Output:  4000      E8h
         3001      12h                 4001      DEh
         3002      22h                 4002      37h
         3003      11h                 4003      01h
```

PROGRAM- 12: Write an 8086 program to divide two 8-bit numbers i.e. dividend and divisor which are present in the memory locations 3000h and 3001h respectively. Store the result in memory locations 4000h and 4001h i.e. in 4000h store the quotient and in 4001h store the remainder.

Algorithm:
1.Initialize the source pointer & destination pointer with a memory location
2.Move the dividend stored in source pointer location in lower byte of Acc register
3.Increment the source pointer memory location
4.Move the divisor stored in source pointer location in lower byte of other register
5.Divide the data of both registers
6.Store the lower byte of the result in destination location, increment the destination pointer & store the upper byte of the result in destination location
7.End the program

Source Code:

```
Output 2500ad          ; Linker version command
Org 2000h              ; Starting address
Mov SI, 3000h          ; Initialize the source pointer
Mov DI, 4000h          ; Initialize the destination pointer
Mov AX, 0000h          ; Clear AX reg
Mov BX, 0000h          ; Clear BX reg
Mov AL, [SI]           ; Move dividend to AL reg
Inc SI                 ; Increment source pointer
Mov BL, [SI]           ; Move divisor to BL reg
Div BL                 ; Divide
Mov [DI], AL           ; Store quotient in dstn loc
Inc DI                 ; Increment destination pointer
Mov [DI], AH           ; Store remainder in destination loc
Int 3                  ; Halt
```

Result:

```
Input:  3000     ABh      Output:  4000    09h
        3001     12h               4001    09h
```

PROGRAM- 13: Write an 8086 program to divide two 16-bit numbers i.e. dividend in 3000h and 3001h and divisor in 3002h and 3003h respectively. Store the result in memory locations 4000h to 4003h i.e. store quotient in the memory locations 4000h and 4001h and remainder in 4002h and 4003h.

Algorithm:
1.Initialize the source pointer & destination pointer with a memory location
2.Move the dividend stored in source pointer location in Acc register
3.Increment the source pointer memory location
4.Move the divisor stored in source pointer location in other register
5.Divide the data of both registers
6.Store the lower word of the result in destination location, increment the destination pointer & store the upper word of the result in destination location
7.End the program

Source Code:

```
Output 2500ad          ; Linker version command
Org 2000h              ; Starting address
Mov SI, 3000h          ; Initialize the source pointer
Mov DI, 4000h          ; Initialize the destination pointer
Mov AX, 0000h          ; Clear AX reg
Mov BX, 0000h          ; Clear BX reg
Mov DX,0000h           ; Clear DX reg
Mov AL, [SI]           ; Move dividend to AX reg
Inc SI
Mov AH, [SI]
Inc SI                 ; Increment source pointer
Mov BL, [SI]           ; Move divisor to BX reg
Inc SI
Mov BH, [SI]
Div BX                 ; Divide
Mov [DI], AL           ; Store quotient in dstn loc
Inc DI
Mov [DI], AH
```

```
Inc DI              ; Increment destination pointer
Mov [DI], DL        ; Store remainder in dstn loc
Inc DI
Mov [DI], DH
Int 3               ; Halt
```

Result:

```
Input:  3000    76h          Output:  4000    08h
        3001    98h                   4001    00h
        3002    34h                   4002    D6h
        3003    12h                   4003    06h
```

EXPERIMENT- 4
Multi byte binary and BCD addition and subtraction

Program 14: Write an ALP for adding two multi byte binary numbers. The two strings of binary numbers starts from memory location 3100h & 3110h respectively. The result is stored from memory location 3120h onwards.

Algorithm:
1.Initialize the source pointers & destination pointer & a counter
2.Clear Acc register
3.Move the data stored in first source location to Acc register
4.Add with carry the data in Acc register with the data in second source pointer location
5.Place the sum in destination location
6.Increment the pointers & repeat the process till the count becomes zero
7.End the program

Source Code:

```
     Output 2500ad          ; Linker version command
     Org 2000h              ; Starting address
     Mov SI,3100h           ; Initialize the first source pointer
     Mov DI,3110h           ; Initialize the second source pointer
     Mov BX,3120h           ; Initialize the result location
     Mov CX,0004h           ; Initialize the count
     Xor AX,AX              ; Clear AX reg
BACK:Mov AL,[SI]            ; Move the data from SI to AL reg
     Adc AL,[DI]            ; Add with cy the data in DI with AL
     Mov [BX],AL            ; Store the sum in result location
     Inc SI                 ; Increment SI
     Inc DI                 ; Increment DI
     Inc BX                 ; Increment BX
     Loop BACK              ; Repeat the process till CX=0
     Int 3                  ; Halt
     Org 3100h
     Db 76h, 14h, 29h, 11h
     Org 3110h
     Db 14h, 27h, 10h, 45h
```

Result:

3120	8Ah
3121	3Bh
3122	39h
3123	56h

Program 15: Write an ALP for subtracting two multi byte binary numbers. The two strings of binary numbers starts from memory location 3100h & 3110h respectively. The result is stored from memory location 3120h onwards.

Algorithm:

1. Initialize the source pointers & destination pointer & a counter
2. Clear Acc register
3. Move the data stored in first source location to Acc register
4. Subtract with barrow the data in Acc register with the data in second source pointer location
5. Place the difference in destination location
6. Increment the pointers & repeat the process till the count becomes zero
7. End the program

Source Code:

```
      Output 2500ad         ; Linker version command
      Org 2000h             ; Starting address
      Mov SI,3100h          ; Initialize the first source pointer
      Mov DI,3110h          ; Initialize the second source pointer
      Mov BX,3120h          ; Initialize the result location
      Mov CX,0004h          ; Initialize the count
      Xor AX,AX             ; Clear AX reg
BACK:Mov AL,[SI]            ; Move the data from SI to AL reg
      Sbb AL,[DI]           ; Sub with cy the data in DI with AL
      Mov [BX],AL           ; Store the difference in result location
      Inc SI                ; Increment SI
      Inc DI                ; Increment DI
      Inc BX                ; Increment BX
      Loop BACK             ; Repeat the process till CX=0
      Int 3                 ; Halt
      Org 3100h
      Db 11h, 29h, 13h, 56h
      Org 3110h
      Db 80h, 16h, 41h, 31h
```

Result:

3120	91h
3121	12h
3122	D2h
3123	24h

Program 16: Write an ALP for adding two multi byte BCD numbers. The two strings of BCD numbers starts from memory location 3100h & 3110h respectively. The result is stored from memory location 3120h onwards.

Algorithm:
1.Initialize the source pointers & destination pointer & a counter
2.Clear Acc register
3.Move the data stored in first source location to Acc register
4.Add with carry the data in Acc register with the data in second source pointer location
5.Convert the binary value into its BCD equivalent
6.Place the sum in destination location
7.Increment the pointers & repeat the process till the count becomes zero
8.End the program

Source Code:

```
        Output 2500ad        ; Linker version command
        Org 2000h            ; Starting address
        Mov SI,3100h         ; Initialize the first source pointer
        Mov DI,3110h         ; Initialize the second source pointer
        Mov BX,3120h         ; Initialize the result location
        Mov CX,0004h         ; Initialize the count
        Xor AX,AX            ; Clear AX reg
BACK:Mov AL,[SI]             ; Move the data from SI to AL reg
        Adc AL,[DI]          ; Add with cy the data in DI with AL
        Daa                  ; Dacimal adjust after addition
        Mov [BX],AL          ; Store the sum in result location
        Inc SI               ; Increment SI
        Inc DI               ; Increment DI
        Inc BX               ; Increment BX
        Loop BACK            ; Repeat the process till CX=0
        Int 3                ; Halt
        Org 3100h
        Db 76h, 14h, 29h, 11h
        Org 3110h
        Db 14h, 27h, 10h, 45h
```

```
Result:   3120    90d
          3121    41d
          3122    39d
          3123    56d
```

Program 17: Write an ALP for subtracting two multi byte BCD numbers. The two strings of BCD numbers starts from memory location 3100h & 3110h respectively. The result is stored from memory location 3120h onwards.

Algorithm:

1. Initialize the source pointers & destination pointer & a counter
2. Clear Acc register
3. Move the data stored in first source location to Acc register
4. Subtract with barrow the data in Acc register with the data in second source pointer location
5. Convert the binary value into its BCD equivalent
6. Place the difference in destination location
7. Increment the pointers & repeat the process till the count becomes zero
8. End the program

Source Code:

```
      Output 2500ad       ; Linker version command
      Org 2000h           ; Starting address
      Mov SI,3100h        ; Initialize the first source pointer
      Mov DI,3110h        ; Initialize the second source pointer
      Mov BX,3120h        ; Initialize the result location
      Mov CX,0004h        ; Initialize the count
      Xor AX,AX           ; Clear AX reg
BACK:Mov AL,[SI]          ; Move the data from SI to AL reg
      Sbb AL,[DI]         ; Sub with cy the data in DI with AL
      Das                 ; Dacimal adjust after subtraction
      Mov [BX],AL         ; Store the difference in result location
      Inc SI              ; Increment SI
      Inc DI              ; Increment DI
      Inc BX              ; Increment BX
      Loop BACK           ; Repeat the process till CX=0
      Int 3               ; Halt
      Org 3100h
      Db 11h, 29h, 13h, 56h
      Org 3110h
      Db 80h, 16h, 41h, 31h
```

Result:

3120	31d
3121	12d
3122	72d
3123	24d

Experiment No: 5
Code Conversions

Program 18: Write an ALP to convert the contents of memory location 3100h into an ASCII character. The 3100h location contains a single Hex digit (4 MSB is zero). Store the ASCII equivalent in memory location 3200h.

```
Algorithm:
1. Initialize the source pointer & destination pointer registers
2. Compare the data with 0Ah
3. If it is <0Ah, add 30h else add 37h
4. Store the result in destination pointer register
5. End the program
```

Source Code:

```
       Output 2500ad          ; Linker version command
       Org 2000h              ; Starting address
       Mov SI,3100h           ; Initialize the source pointer
       Mov DI,3200h           ; Initialize the destination pointer
       Mov AX,0000h           ; Clear AX reg
       Mov AL,[SI]            ; Move the hex data from SI to AL reg
       Cmp AL,0Ah             ; Compare it with 0AH
       Jc AHEAD               ; Jump if cy=1 to AHEAD
       Add AL,'A'-'9'-1  (or)  Add AL,07h ; Add 07H to AL reg
AHEAD:Add AL,30h              ; Add 30H to AL reg
       Mov [DI],AL            ; Store the result in dest ptr location
       Int 3                  ; Halt
       Org 3100h
       Db 09h
```

Result: 3200h 39h

Program 19: Write an ALP to convert the contents of memory location 3100h from ASCII to an equivalent Hexadecimal number. Place the result at memory location 3200h.

Algorithm:
1. Initialize the source pointer & destination pointer registers
2. Subtract 30h from the input data
3. Compare the value with 09h
3. If it is >09h, subtract 07h else do nothing
4. Store the result in destination pointer register
5. End the program

Source Code:

```
             Output 2500ad          ; Linker version command
             Org 2000h              ; Starting address
             Mov SI,3100h           ; Initialize the source pointer
             Mov DI,3200h           ; Initialize the destination pointer
             Mov AX,0000h           ; Clear AX reg
             Mov AL,[SI]            ; Move the hex data from SI to AL reg
             Sub AL,30h             ; Subtract 30H from AL reg
             Cmp AL,09h             ; Compare value of AL reg with 09H
             Jg AHEAD               ; Jump if AL>09H to AHEAD
             Jmp DONE               ; Jump DONE
       AHEAD:Sub AL,07h             ; Subtract 07H from AL reg
        DONE:Mov [DI],AL            ; Store the result in dest ptr location
             Int 3                  ; Halt
             Org 3100h
             Db 43h
```

Result: 3200h 0Ch

Program 20: Write an ALP to convert 2 digit packed BCD number into its Binary equivalent number. Packed 2 digit number is stored in memory location 3100h & place the result at memory location 3200h.

Algorithm:

1. Initialize the source pointer & destination pointer registers
2. Split the packed BCD into two unpacked BCD digits BCD1 & BCD2
3. Multiply BCD2 by 10 i.e., 0ah
4. Add BCD1 to the answer in step 2
5. Store the result in destination location
6. End the program

Source Code:

```
Output 2500ad          ; Linker version command
Org 2000h              ; Starting address
Mov SI,3100h           ; Initialize the source pointer
Mov DI,3200h           ; Initialize the destination pointer
Mov AL,[SI]            ; Move the data from SI to AL reg
And AL,0Fh             ; Mask the upper nibble of AL reg
Mov DL,AL              ; Save BCD1 in DL reg
Mov AL,[SI]            ; Move the data from SI to AL reg again
And AL,0F0h            ; Mask the lower nibble of AL reg
Mov CL,04h             ; Initialize the count
Ror AL,CL              ; Rotate right AL CL times to get BCD2
Mov BL,0Ah             ; Move 0AH to BL reg
Mul BL                 ; Multiply it with BCD2
Add AL,DL              ; Add it with BCD1
Mov [DI],AL            ; Store the result in dest ptr location
Int 3                  ; Halt
Org 3100h
Db 69h
```

Result: 3200 45h

Program 21: Write an ALP to convert Binary number into its 2 digit packed BCD number. The binary number is stored in memory location 2100h & place the result at memory location 2200h.

Algorithm:
1. Initialize the source pointer & destination pointer registers
2. Divide the value with 0Ah
3. Rotate the quotient 4 times & add it with remainder
4. Store the result in destination location

Source Code:

```
Output 2500ad          ; Linker version command
Org 2000h              ; Starting address
Mov SI,2100h           ; Initialize the source pointer
Mov DI,2200h           ; Initialize the destination pointer
Mov AX,0000h           ; Clear AX reg
Mov AL,[SI]            ; Move the data from SI to AL reg
Mov BL, 0Ah            ; Move 0AH to BL reg
Div BL                 ; Divide AL with BL
Mov CL,04h             ; Initialize the count
Rol AL,CL              ; Rotate left quotient CL times
Or AL,AH               ; Add it with remainder
Mov [DI],AL            ; Store the result in dest ptr location
Int 3                  ; Halt
Org 2100h
Db 45h
```

Result: 2200 69d

Program 22: Write an ALP to convert decimal number into its equivalent 7-segment conversion using XLAT instruction. The 7-segment codes are stored in memory as a lookup table starting from 2100h (use common cathode codes) for 7-segment display.

Algorithm:

1. Initialize the source pointer & destination pointer registers
2. Save the source location address in BX register
3. Take the input in AL register
4. Convert the decimal value into 7-segment code
5. Store the result in destination location
6. End the program

Source Code:

```
Output 2500ad          ; Linker version command
 Org 2000h             ; Starting address
 Mov SI,2100h          ; Initialize the source pointer
 Mov DI,2200h          ; Initialize the destination pointer
 Mov BX,SI             ; Copy the source address in BX reg
 Mov AL,05h            ; Move the input data in AL reg
 Xlat                  ; Translate byte in AL from lookup table
 Mov [DI],AL           ; Store the result in dest ptr location
 Int 3                 ; Halt
 Org 2100h
 Db 3fh, 06h, 5bh, 4fh, 66h, 6dh, 7dh,
 07h, 7fh, 6fh
```

Result: 2200 6dh

Program 23: Write an ALP to convert temperature from degree centigrade into degree Fahrenheit using C=5/9*(F-32).
F=9C/5+32 ➔ F=9C/5+20h

Algorithm:
1. Initialize the source pointer & destination pointer registers
2. Multiply the input with 09h
3. Divide it with 05h & add with 20h
4. Store the result in destination location
5. End the program

Source Code:

```
Output 2500ad          ; Linker version command
Org 2000h              ; Starting address
Mov SI,2100h           ; Initialize the source pointer
Mov DI,2200h           ; Initialize the destination pointer
Mov AL,[SI]            ; Move the data from SI to AL reg
Mov BL,09h             ; Move 09H to BL reg
Mul BL                 ; Multiply AL with BL regs
Mov CL,05h             ; Initialize CL reg
Div CL                 ; Divide AL with CL reg
Add AL,20h             ; Add it with 20H
Mov [DI],AL            ; Store the result in dest ptr location
Int 3                  ; Halt
Org 2100h
Db 05h
```

Result: 2200 29h

EXPERIMENT- 6
String Searching and Sorting

Program 24: Copy a list of characters from one memory location to other using string manipulation instructions. The Source string starts from 2100h onwards & the destination string begins from 2200h.

```
Algorithm:
1. Initialize the source pointer, destination pointer & counter registers
2. Clear direction flag
3. Move string byte from source location to destination location
4. Repeat the process till the count becomes zero
5. End the program
```

Source Code:

```
Output 2500ad          ; Linker version command
Org 2000h              ; Starting address
Mov CX, 0Bh            ; Initialize the count
Mov SI, 2100h          ; Initialize the source pointer
Mov DI, 2200h          ; Initialize the destination pointer
Cld                    ; Clear direction flag
Rep                    ; Repeat till CX=00H
Movsb                  ; Move string byte from scr ptr to dest ptr
Int 3                  ; Halt
Org 2100h
Db ‘ELECTRONICS’
```

Result 2200: ’ELECTRONICS’

Command to display a string → D 2200, 220B

Program 25: To search for a character in the given string & store 00h if it is present in memory location 2100h.

Algorithm:
1. Initialize the source pointer, destination pointer & counter registers
2. Move the string byte which has to be searched in AL register
3. Clear direction flag
3. Scan string byte from source location with the value in AL register
4. Repeat the process till the value becomes equal or count becomes zero
5. Store the result in destination location
6. End the program

Source Code:

```
        Output 2500ad         ; Linker version command
        Org 2000h             ; Starting address
        Mov SI,2100h          ; Initialize the destination pointer
        Mov DI,2200h          ; Initialize the source pointer
        Mov CX,000Ah          ; Initialize the count
        Mov AL,45h (Ascii equivalent of ‘E’); Move 45H to AL reg
        Cld                   ; Clear direction flag
        Repne                 ; Repeat if AL=!45H
        Scasb                 ; Scan string byte
        Jne Exit              ; Jump if AL!=45H when CX=00H
        Mov AL,00h  or  Mov byte ptr [SI],00h in S.M; Move 00H in AL reg
        Mov [SI],AL           ; Store the result in dest ptr location
  Exit: Int 3                 ; Halt
        Org 2200h
        Db ‘ABCDEFGHIJ’
```

Result: 2100 00h

Program 26: Write an 8086 ALP that exchanges two blocks of data stored in memory using the string instructions of 8086. This program exchanges 0F bytes from 0:3000h and 0:3200h onwards.

Algorithm:
1. Initialize the source pointer, destination pointer & counter registers
2. Clear direction flag
3. Move string byte from source location to destination location
4. Repeat the process till the count becomes zero to move first string in dummy location
5. Repeat the steps 1-4 to move second string in the location of first string
6. Repeat the steps 1-4 to move first string from dummy location to second string location
5. End the program

Source Code:

```
Output 2500ad          ; Linker version command
Org 2000h              ; Starting address
Mov BX,3000h           ; Move first string address to BX reg
Mov DX,3200h           ; Move second string address to DX reg
Mov AX,3500h           ; Move dummy string address to AX reg
Mov SI,BX              ; Initialize the source pointer
Mov DI,AX              ; Initialize the destination pointer
Mov CX,000Fh           ; Initialize the count
Cld                    ; Clear direction flag
Rep                    ; Repeat till CX=00H
Movsb                  ; Move string byte from scr ptr to dest ptr
Mov SI,DX              ; Initialize the source pointer
Mov DI,BX              ; Initialize the destination pointer
Mov CX,000Fh           ; Initialize the count
Cld                    ; Clear direction flag
Rep                    ; Repeat till CX=00H
Movsb                  ; Move string byte from scr ptr to dest ptr
Mov SI,AX              ; Initialize the source pointer
Mov DI,DX              ; Initialize the destination pointer
Mov CX,000Fh        ; Initialize the count
Cld                 ; Clear direction flag
Rep                 ; Repeat till CX=00H
Movsb               ; Move string byte from scr ptr to dest ptr
Int 3               ; Halt
Org 3000h
Db ‘ABCD…………..O’
Org 3200h
Db ‘QRST……………E’
```

```
D 3000,300E
D 3200,320E
```

Result:

```
3000        ‘QRSTUVWXYZABCDE’
3200        ‘ABCDEFGHIJKLMNO’
```

Title: - Ascending order using bubble sort, a set of unsigned numbers.

Approach Methodology:

Let us say, we want to arrange the numbers 34,78,56,47 in ascending order using bubble sort algorithm.

We compare first two elements i.e., 34 and 78. Since 34<78, the two numbers are in proper order.

Next, compare the 2nd and 3rd elements i.e., 78 and 56, since they are not in order, the elements are interchanged. Now the elements of the vector appear as 34,56,78,47.

Next we compare the 3rd and last element i.e. 78 and 47. Since they are not in order they are interchanged. Thus the largest element of the unsorted vector is placed in the correct position. Now, the elements of the vector appear as 34, 56, 47 | 78 where the elements after '|' are sorted in ascending order.

Therefore, in 3 passes the sorting is completed.
After 2nd pass, the elements of the vector are 34, 47 | 56, 78.
After 3rd pass, the elements of the vector are 34 | 47, 56, 78.

In General it needs (n-1) passes when 'n' elements are to be sorted.

Program 27: Write an ALP to sort an array of unsigned binary numbers in ascending order. The array begins at memory location 2100h.

Algorithm:

Description: Here A is an unsorted array having N elements.

```
1. Repeat For J = 1 to N
2.     Repeat For K = 1 to N-J
3.         If (A[K] > A[K+1]) Then
4.             Interchange A[K] and A[K+1]
                           [End of If]
                    [End of Step 2 For Loop]
             [End of Step 1 For Loop]
5. End the program
```

Source Code:

```
                  Output 2500ad         ; Linker version command
                  Org 2000h             ; Starting address
                  Mov BX,0Ah            ; Move the no of elements in BX reg
                  Dec BX                ; Decrement BX reg
         OUTLUP:Mov CX,BX               ; Place the count in CX reg
                  Mov SI,2100h          ; Initialize the source pointer
          INLUP:Mov AL,[SI]             ; Move the data from SI to AL reg
                  Inc SI                ; Increment SI
                  Cmp AL,[SI]           ; Compare the data in SI with AL reg
                  Jb NEXT               ; Jump if AL<[SI] to NEXT
                  Xchg AL,[SI]          ; Exchange the data
                  Dec SI                ; Decrement SI
                  Mov [SI],AL           ; Move AL data to SI location
                  Inc SI                ; Increment SI
           NEXT:Loop INLUP              ; Repeat the INLUP till CX=0
                  Dec BX                ; Decrement BX reg
                  Jnz OUTLUP            ; Jump if BX!=0 to OUTLUP
                  Int 3                 ; Halt
                  Org 2100h
                  Db 10h,42h,11h,05h,01h,79h,34h,67h,02h,12h
```

Result:	2100	01h
	2101	02h
	2102	05h
	2103	10h
	2104	11h
	2105	12h
	2106	34h
	2107	42h
	2108	67h
	2109	79h

Program 28: Write an ALP to sort an array of unsigned binary numbers in descending order. The array begins at memory location 2100h.

Algorithm:

Description: Here A is an unsorted array having N elements.

```
1. Repeat For J = 1 to N
2.     Repeat For K = 1 to N-J
3.         If (A[K] < A[K+1]) Then
4.             Interchange A[K] and A[K+1]
                             [End of If]
                      [End of Step 2 For Loop]
               [End of Step 1 For Loop]
5. End the program
```

Source Code:

```
                  Output 2500ad         ; Linker version command
                  Org 2000h             ; Starting address
                  Mov BX,0Ah            ; Move the no of elements in BX reg
                  Dec BX                ; Decrement BX reg
          OUTLUP:Mov CX,BX              ; Place the count in CX reg
                  Mov SI,2100h          ; Initialize the source pointer
           INLUP:Mov AL,[SI]            ; Move the data from SI to AL reg
                  Inc SI                ; Increment SI
                  Cmp AL,[SI]           ; Compare the data in SI with AL reg
                  Jg NEXT               ; Jump if AL>[SI] to NEXT
                  Xchg AL,[SI]          ; Exchange the data
                  Dec SI                ; Decrement SI
                  Mov [SI],AL           ; Move AL data to SI location
                  Inc SI                ; Increment SI
            NEXT:Loop INLUP             ; Repeat the INLUP till CX=0
                  Dec BX                ; Decrement BX reg
                  Jnz OUTLUP            ; Jump if BX!=0 to OUTLUP
                  Int 3                 ; Halt
                  Org 2100h
                  Db 10h,42h,11h,05h,01h,79h,34h,67h,02h,12h
```

Result:

2100	79h
2101	67h
2102	42h
2103	34h
2104	12h
2105	11h
2106	10h
2107	05h
2108	02h
2109	01h

Experiment No: 7
Generation of Waveforms Using DAC Interface Module

AIM: To write and execute programs in 8086 for interfacing a DAC module with ESA 86/88E Microprocessor trainer kit.

APPARATUS:

1. Microprocessor kit

2. Dual channel DAC interface unit

3.26 pin connector cable

4. Power supply unit

5. CRO

DISCRIPTION: To use DAC, initialize 8255 for mode 0 operation with a port A and port B as output. Output data on the appropriate ports and observe the output waveform at X out and Y out of DAC using CRO.

The 16 bit port addresses of 8255 PPI low map on even bus available at J4 connector are:

```
PORT A      EQU      FFE0H

PORT B      EQU      FFE2H

PORT C      EQU      FFE4H

CWR         EQU      FFE6H
```

Hardware Details of DAC Interface Circuit:

The Dual DAC interface can be used to generate different waveforms using ESA 86/88E Microprocessor kit.

There are two 8 bit digital to analog converters provided based on DAC 0800.
The digital inputs to these DACs are provided through PORT A & PORT B of 8255 used as output ports.

The analog outputs from the DACs are given to OPAMPS (µ741) which act as current to voltage converters.

The outputs from the DACs vary between 0 to 5V corresponding values between 00 to FF hex.

Different waveforms can be observed at the OPAMP outputs depending upon the digital input patterns.

Description of the Circuit:

The Dual DAC circuit uses only 17 lines from 26 pin connector cable.

PORT A & PORT B of 8255 PPI are used as output ports. The digital inputs to DACs are connected to the inverting inputs of OPAMPs (µ741).

The outputs from the OPAMPs are connected to points marked X out & Y out at which the waveforms are observed on CRO.

PORT A is used to control X out & PORT B controls Y out.

Program 29: Write an 8086 ALP to generate up-going saw-tooth waveform.

Algorithm:
1. Initialize the 8255 in mode 0 & all ports as output ports
2. Initialize AL with 00h
3. Send the data to the ports A & B through AL register
4. Increment AL
5. Repeat the steps 3 & 4 for the generation of continuous waveform

Source Code:

```
          Output 2500ad           ; Linker version command
          Org 2000h               ; Starting address
          Mov AL,80h              ; Initialize 8255 in mode 0
          Mov DX,0FFE6h           ; & all the ports
          Out DX,AL               ; as output ports
          Mov AL,00h              ; Move 00H to AL reg
     BACK:Mov DX,0FFE0h           ; Move port A addr to DX reg
          Out DX,AL               ; Send AL value through port A
          Mov DX,0FFE2h           ; Move port A addr to DX reg
          Out DX,AL               ; Send AL value through port B
          Inc AL                  ; Increment AL reg
          Jmp BACK   or    Jmp short BACK for S.M; Repeat
```

Result: The output waveform is observed on CRO & amplitude and time period is measured.

Program 30: Write an 8086 ALP to generate down-going saw-tooth waveform.

Algorithm:
1. Initialize the 8255 in mode 0 & all ports as output ports
2. Initialize AL with FFh
3. Send the data to the ports A & B through AL register
4. Decrement AL
5. Repeat the steps 3 & 4 for the generation of continuous waveform

Source Code:

```
      Output 2500ad       ; Linker version command
      Org 2000h           ; Starting address
      Mov AL,80h          ; Initialize 8255 in mode 0
      Mov DX,0ffe6h       ; & all the ports
      Out DX,AL           ; as output ports
      Mov AL,0FFh         ; Move FFH to AL reg
BACK:Mov DX,0FFE0h        ; Move port A addr to DX reg
      Out DX,AL           ; Send AL value through port A
      Mov DX,0FFE2h       ; Move port A addr to DX reg
      Out DX,AL           ; Send AL value through port B
      Dec AL              ; Decrement AL reg
      Jmp BACK            ; Repeat
```

Result: The output waveform is observed on CRO & amplitude and time period is measured.

Program 31: Write an 8086 ALP to generate triangular waveform.

Algorithm:
1. Initialize the 8255 in mode 0 & all ports as output ports
2. Initialize AL with 00h
3. Send the data to the ports A & B through AL register
4. Increment AL
5. Repeat the steps 3 & 4 for the generation of up-going saw-tooth waveform
6. Repeat the steps 2-5 with AL = FFh & decrementing AL for the generation of down-going saw-tooth
7. Repeat the steps 2-6 for the generation of continuous waveform

Source Code:

```
        Output 2500ad         ; Linker version command
        Org 2000h             ; Starting address
        Mov AL,80h            ; Initialize 8255 in mode 0
        Mov DX,0FFE6h         ; & all the ports
        Out DX,AL             ; as output ports
        Mov AL,00h            ; Move 00H to AL reg
     UP:Mov DX,0FFE0h         ; Move port A addr to DX reg
        Out DX,AL             ; Send AL value through port A
        Mov DX,0FFE2h         ; Move port B addr to DX reg
        Out DX,AL             ; Send AL value through port B
        Inc AL                ; Increment AL
        Cmp AL,0FFh           ; Compare value of AL with FFH
        Jb UP                 ; Jump if AL<FFH to UP
   DOWN:Mov DX,0FFE0h         ; Move port A addr to DX reg
        Out DX,AL             ; Send AL value through port A
        Mov DX,0FFE2h         ; Move port B addr to DX reg
        Out DX,AL             ; Send AL value through port B
        Dec AL                ; Decrement AL reg
        Cmp AL,00h            ; Compare value of AL with 00H
        Ja DOWN               ; Jump if AL>00H to DOWN
        Jmp UP                ; Repeat
```

Result: The output waveform is observed on CRO & amplitude and time period is measured.

Program 32: Write an 8086 ALP to generate symmetrical square waveform.

Algorithm:
1. Initialize the 8255 in mode 0 & all ports as output ports
2. Initialize AL with 00h
3. Send the data to the port A through AL register & provide the delay
4. Make AL = FFh, send it through port A & provide the delay
5. Repeat the steps 2-4 for the generation of continuous waveform

Source Code:

```
             Output 2500ad          ; Linker version command
             Org 2000h              ; Starting address
             Mov AL,80h             ; Initialize 8255 in mode 0
             Mov DX,0FFE6h          ; & all the ports
             Out DX,AL              ; as output ports
        BACK:Mov AL,00h             ; Move 00H to AL reg
             Mov DX,0FFE0h          ; Move port A addr to DX reg
             Out DX,AL              ; Send AL value through port A
             Mov CX,0FFh            ; Initialize the count
        DLY1:Loop DLY1              ; Provide delay till CX=0
             Mov AL,0FFh            ; Move FFH to AL reg
             Mov DX,0FFE0h          ; Move port A addr to DX reg
             Out DX,AL              ; Send AL value through port A
             Mov CX,0FFh            ; Initialize the same count
        DLY2:Loop DLY2              ; Provide delay till CX=0
             Jmp BACK               ; Repeat
```

Result: The output waveform is observed on CRO & amplitude and time period is measured.

Program 33: Write an 8086 ALP to generate up-going staircase waveform containing 5 steps.

Algorithm:
1. Initialize the 8255 in mode 0 & all ports as output ports
2. Initialize AL with 00h
3. Add 33h to AL & send the data to the port A through AL register & provide the delay
4. Compare AL with FFh & Repeat the steps 2 & 3 for the generation of continuous waveform

Source Code:

```
      Output 2500ad        ; Linker version command
      Org 2000h            ; Starting address
      Mov AL,80h           ; Initialize 8255 in mode 0
      Mov DX,0FFE6h        ; & all the ports
      Out DX,AL            ; as output ports
      Mov AL,00h           ; Move 00H to AL reg
  RPT:Add AL,33h           ; Add 33H to AL reg
      Mov DX,0FFE0h        ; Move port A addr to DX reg
      Out DX,AL            ; Send AL value through port A
      Mov CX,0FFh          ; Initialize the count
 DLY1:Loop DLY1            ; Provide delay till CX=00H
      Cmp AL,0FFh          ; Compare AL with FFH
      Jnz RPT              ; Jump if AL!=00H to RPT
      Inc AL               ; Increment AL reg
      Mov DX,0FFE0h        ; Move port A addr to DX reg
      Out DX,AL            ; Send AL value through port A
      Mov CX,0FFh          ; Initialize the count
 DLY2:Loop DLY2            ; Provide delay till CX=00H
      Jmp RPT              ; Repeat
```

Result: The output waveform is observed on CRO & amplitude and time period is measured.

Program 34: Write an 8086 ALP to generate down-going staircase waveform containing 5 steps.

Algorithm:
1. Initialize the 8255 in mode 0 & all ports as output ports
2. Initialize AL with FFh
3. Subtract 33h from AL & send the data to the port A through AL register & provide the delay
4. Compare AL with 00h & Repeat the steps 2 & 3 for the generation of continuous waveform

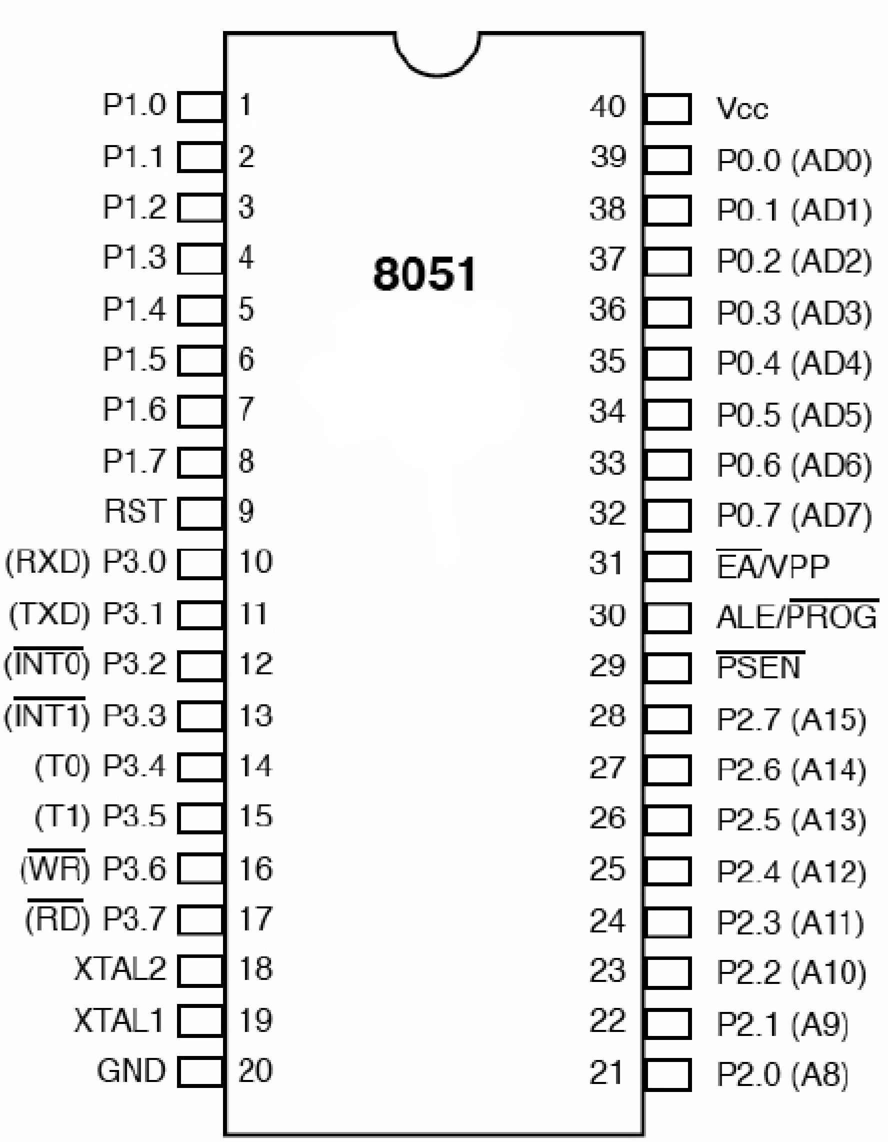
Source Code:

```
          Output 2500ad        ; Linker version command
          Org 2000h            ; Starting address
          Mov AL,80h           ; Initialize 8255 in mode 0
          Mov DX,0FFE6h        ; & all the ports
          Out DX,AL            ; as output ports
          Mov AL,0FFh          ; Move FFH to AL reg
      RPT:Sub AL,33h           ; Subtract 33H from AL reg
          Mov DX,0FFE0h        ; Move port A addr to DX reg
          Out DX,AL            ; Send AL value through port A
          Mov CX,0FFh          ; Initialize the count
     DLY1:Loop DLY1            ; Provide delay till CX=00H
          Cmp AL,00h           ; Compare AL with 00H
          Jnz RPT              ; Jump if AL!=00H to RPT
          Dec AL               ; Decrement AL reg
          Mov DX,0FFE0h        ; Move port A addr to DX reg
         Out DX,AL             ; Send AL value through port A
         Mov CX,0FFh           ; Initialize the count
    DLY2:Loop DLY2             ; Provide delay till CX=00H
         Jmp RPT               ; Repeat
```

Result: The output waveform is observed on CRO & amplitude and time period is measured.

PART B

[Experiments on assembly language programming for 8051 using Assembler]



8051 Microcontroller pin diagram

8051 Instruction Set

ARITHMETIC OPERATIONS:

ADD A,Rn	Add register to Accumulator
ADD A,direct	Add direct byte to Accumulator
ADD A,@Ri	Add indirect RAM to Accumulator
ADD A,#data	Add immediate data to Accumulator
ADDC A,Rn	Add register to Accumulator with Carry
ADDC A,direct	Add direct byte to Accumulator with Carry
ADDC A,@Ri	Add indirect RAM to Accumulator with Carry
ADDC A,#data	Add immediate data to Acc with Carry
SUBB A,Rn	Subtract Register from Acc with borrow
SUBB A,direct	Subtract direct byte from Acc with borrow
SUBB A,@Ri	Subtract indirect RAM from ACC with borrow
SUBB A,#data	Subtract immediate data from Acc with borrow
INC A	Increment Accumulator
INC Rn	Increment register
INC direct	Increment direct byte
INC @Ri	Increment direct RAM
DEC A	Decrement Accumulator
DEC Rn	Decrement Register
DEC direct	Decrement direct byte
DEC @Ri	Decrement indirect RAM
INC DPTR	Increment Data Pointer
MUL AB	Multiply A & B
DIV AB	Divide A by B
DA A	Decimal Adjust Accumulator

LOGICAL OPERATIONS:

ANL A,Rn	AND Register to Accumulator
ANL A,direct	AND direct byte to Accumulator
ANL A,@Ri	AND indirect RAM to Accumulator
ANL A,#data	AND immediate data to Accumulator
ANL direct,A	AND Accumulator to direct byte
ANL direct,#data	AND immediate data to direct byte
ORL A,Rn	OR register to Accumulator
ORL A,direct	OR direct byte to Accumulator
ORL A,@Ri	OR indirect RAM to Accumulator
ORL A,#data	OR immediate data to Accumulator
ORL direct,A	OR Accumulator to direct byte
ORL direct,#data	OR immediate data to direct byte
XRL A,Rn	Exclusive-OR register to Accumulator
XRL A,direct	Exclusive-OR direct byte to Accumulator
XRL A,@Ri	Exclusive-OR indirect RAM to Accumulator

XRL A,#data	Exclusive-OR immediate data to Accumulator
XRL direct,A	Exclusive-OR Accumulator to direct byte
XRL direct,#data	Exclusive-OR immediate data to direct byte
CLR A	Clear Accumulator
CPL A	Complement Accumulator
RL A	Rotate Accumulator Left
RLC A	Rotate Accumulator Left through the Carry
RR A	Rotate Accumulator Right
RRC A	Rotate Accumulator Right through the Carry
SWAP A	Swap nibbles within the Accumulator

DATA TRANSFER:

MOV A,Rn	Move register to Accumulator
MOV A,direct	Move direct byte to Accumulator
MOV A,@Ri	Move indirect RAM to Accumulator
MOV A,#data	Move immediate data to Accumulator
MOV Rn,A	Move Accumulator to register
MOV Rn,direct	Move direct byte to register
MOV Rn,#data	Move immediate data to register
MOV direct,A	Move Accumulator to direct byte
MOV direct,Rn	Move register to direct byte
MOV direct,direct	Move direct byte to direct
MOV direct,@Ri	Move indirect RAM to direct byte
MOV direct,#data	Move immediate data to direct byte
MOV @Ri,A	Move Accumulator to indirect RAM
MOV @Ri,direct	Move direct byte to indirect RAM
MOV @Ri,#data	Move immediate data to indirect RAM
MOV DPTR,#data16	Load Data Pointer with a 16-bit constant
MOVC A,@A+DPTR	Move Code byte relative to DPTR to Acc
MOVC A,@A+PC	Move Code byte relative to PC to Acc
MOVX A,@Ri	Move External RAM (8- bit addr) to Acc
MOVX A,@DPTR	Move Exernal RAM (16- bit addr) to Acc
MOVX @Ri,A	Move Acc to External RAM (8-bit addr)
MOVX @DPTR,A	Move Acc to External RAM (16-bit addr)
PUSH direct	Push direct byte onto stack
POP direct	Pop direct byte from stack
XCH A,Rn	Exchange register with Accumulator
XCH A,direct	Exchange direct byte with Accumulator
XCH A,@Ri	Exchange indirect RAM with Accumulator
XCHD A,@Ri	Exchange low-order Digit indirect RAM with Acc

BOOLEAN VARIABLE MANIPULATION:

CLR C	Clear Carry
CLR bit	Clear direct bit

SETB C	Set Carry
SETB bit	Set direct bit
CPL C	Complement Carry
CPL bit	Complement direct bit
ANL C,bit	AND direct bit to CARRY
ANL C,/bit	AND complement of direct bit to Carry
ORL C,bit	OR direct bit to Carry
ORL C,/bit	OR complement of direct bit to Carry
MOV C,bit	Move direct bit to Carry
MOV bit,C	Move Carry to direct bit
JC rel	Jump if Carry is set
JNC rel	Jump if Carry not set
JB bit,rel	Jump if direct Bit is set
JNB bit,rel	Jump if direct Bit is Not set
JBC bit,rel	Jump if direct Bit is set & clear bit

PROGRAM BRANCHING:

ACALL addr11	Absolute Subroutine Call
LCALL addr16	Long Subroutine Call
RET	Return from Subroutine
RETI	Return from interrupt
AJMP addr11	Absolute Jump
LJMP addr16	Long Jump
SJMP rel	Short Jump (relative addr)
JMP @A+DPTR	Jump indirect relative to the DPTR
JZ rel	Jump if Accumulator is Zero
JNZ rel	Jump if Accumulator is Not Zero
CJNE A,direct,rel	Compare direct byte to Acc and Jump if Not Equal
CJNE A,#data,rel	Compare immediate to Acc and Jump if Not Equal
CJNE Rn,#data,rel	Compare immediate to register and Jump if Not Equal
CJNE @Ri,#data,rel	Compare immediate to indirect and Jump if Not Equal
DJNZ Rn,rel	Decrement register and Jump if Not Zero
DJNZ direct,rel	Decrement direct byte and Jump if Not Zero
NOP	No Operation

***Source: Atmel 8051 Microcontrollers Hardware Manual

Experiment No: 8
Familiarity and use of 8051/8031 Microcontroller trainer kit and execution of programs

Main features of ESA 31 (8031 based):

ESA 31 can be operated either from onboard keyboard or from a CRT terminal through its RS 232-C interface.

Keyboard and serial monitor programs support the entry of users program, editing, debug facilities like breakpoints, single stepping & full execution of user programs.

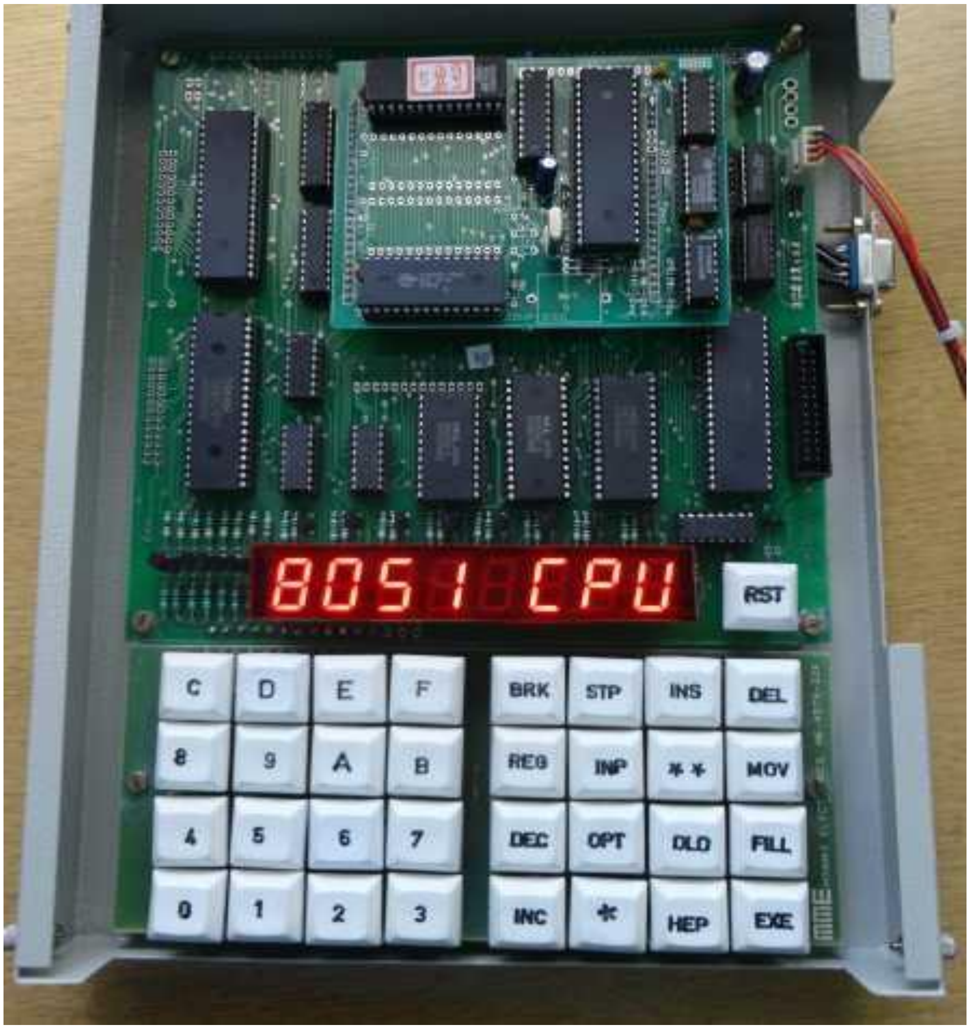
1-pass Assembler can assemble any memory resident assembly language program.

1-pass Dis-assembler disassembles the object code into standard INTEL mnemonics.

Total of **120KB** memory is provided of which **64KB** of memory is program memory and 56KB of memory is data memory.

The monitor of the trainer occupies **32KB** out of **64KB** of program memory.

Standard bus compatible signals available on the bus connector for easy expansion.



Microcontroller 8051 Trainer kit

SPECIFICATIONS:

Microcontroller: 8031/8051 operated at **11.0592 MHz**

Memory: Four 28-pin JEDEC sockets offer 120KB of memory as follows:

32KB of firmware in one 27256 (Program memory)

32KB of SRAM using one 62256 (User program memory)

56KB of SRAM as data memory using two 62256

The memory map is as follows:

DEVICE	**ADDRESS RANGE**	**TYPE OF MEMORY**
27256	0000-7FFF	Program memory
62256	8000-FFFF	User program memory
62256	0000-7FFF	User data memory
62256	8000-FFFF	User data memory

The dip switch settings for either mode of operation are as follows:

For Hexadecimal keypad mode : All switches in OFF position.
For Serial mode : Switches 1 and 4 in ON position.

KEYBOARD MONITOR

In the keyboard mode, the user enters the commands and data by pressing the appropriate keys on the keypad. Responses are displayed by the system on the seven-digit 7-segment LED display.

The RESET key causes a hardware reset and restarts the monitor. The monitor displays the sign-on message **– ESA 51** across the address & data fields of the display.

KEYBOARD & DISPLAY

The display consists of 7 seven segment LED displays, separated into three fields. The leftmost single digit forms the special field. Next four digits form the address field.
Note: Address can be 256B or 64KB max.
Last two digits form the data field.

The 36-key keypad consists of the following group of keys.

A) Hex pad – 16 keys representing hex digits 0 through F.

B) Command Group – 13 command keys.

C) Memory Group – 4 keys (PRGMEM, EXTDATA, BITMEM, INT DATA) to select the type of memory.

D) System Operation keys – RESET, BREAK, and EXEC keys

BREAK can be used to stop the execution without affecting the register contents.

MONITOR COMMANDS

The keyboard monitor is capable of executing fifteen individual commands.

- **EXAMINE/MODIFY MEMORY**

Displays/modifies the contents of a memory location:
Syntax Format:

EXAM MEM {PRG MEM/EXT DATA/BIT MEM/INT DATA}
Addr1 NEXT [[[data] NEXT/PREV]….] EXEC

After pressing EXAM MEM key, enter the type of memory by pressing PRGMEM, EXTDATA, BITMEM, or INTDATA key.

A dot appears at the last digit of the address field indicating that an address entry is required.

Enter the memory address of the byte to be examined. (Memory address is evaluated modulo 64K if it is program memory or data memory and modulo 256 if it is internal data memory or bit memory).

The value is displayed in the address field of the display.

- **EXAMINE/MODIFY REGISTER**

This command is used to examine and optionally modify the contents of some of the 8031/8051's registers.

EXAM REG [reg key] [[Data] NEXT/PREV]….] EXEC

Note: When any of the registers R0-R7 has to be examined, press EXAMREG key and then BITMEM key. Now, press keys 0-7 on the hex keypad which corresponds to registers R0-R7.

The displayed registers contents of R0-R7 is w.r.t the current bank selected.

- **GO COMMAND:**

Is used to transfer control of the system from the monitor to the user's program.

GO [Starting addr] EXEC

To abort execution of user program, press ‘RESET’ key. By doing so, all registers information about user program is lost. In any case, contents of the user portion of the RAM area are not altered by the monitor.

There are two ways to break the user program execution.

a) Set breakpoints at specific addresses and enable them.
b) Press ‘BREAK’ key.

If BREAK key is pressed, control returns to the monitor which saves all the registers and displays the address where the program broke and the data at that address on the display. It displays U on the special field of 7-segment display.

Execution of simple programs using ESA-31 in keyboard mode:

35. Write a program in 8051 to add two 16-bit numbers. The numbers are 3CE7H and 3B8DH. Place the sum in registers R7 and R6; R6 has the lower byte.

Algorithm:
1. Initialize the first lower byte in Acc register & add it with second lower byte
2. Store the lower sum in R6 register
3. Take the first upper byte in Acc register & add it with second upper byte along with carry
4. Store the upper byte of the result in R7 register
4. End the program

Source Code:

Address	**OBJECT Code**	**Mnemonic**
8000	C3	CLR C
8001	74 E7	MOV A, #0E7H
8003	24 8D	ADD A,#8DH
8005	FE	MOV R6,A
8006	74 3C	MOV A,#3CH
8008	34 3B	ADDC A,#3BH
800A	FF	MOV R7,A
800B	80 FE	HERE:SJMP HERE

(Relative address = Target address – PC contents)

Enter the codes using the format given below.

<EXAMMEM><PRGMEM> 8000 <NXT> DATA <NXT>…..<NXT>EXEC

Note: After executing the program using GO<8000>EXEC press ‘BREAK’ key.
Press EXAMREG key twice to check result in registers R6 and R7 respectively.

Format:
<EXAMREG><EXAMREG><BITMEM>6<NEXT><NEXT>

36. Write an 8051 program to multiply two unsigned 8-bit binary numbers. The numbers are stored in memory locations 8050h and 8051h. Store the result in 8060h and 8061h.

Algorithm:
1. Initialize the pointer DPTR & save the multiplier in B reg
2. Again initialize the pointer DPTR & take the multiplicand in Acc register
3. Multiply two 8-bit data
4. Store the result in the respective memory locations
4. End the program

Sample data: (8050)= 41h $(65)_{10}$
(8051)= 08h

Source Code:

```
            ORG 8000H               ; Starting address
            MOV DPTR,#8051h         ; Initialize DPTR
            MOVX A,@DPTR            ; Multiplier in A reg
            MOV 0F0h,A              ; Save Multiplier in B reg
            MOV DPTR,#8050h         ; Initialize DPTR
            MOVX A,@DPTR            ; Multiplicand in A reg
            MUL AB                  ; Multiply
            MOV DPTR,#8060h         ; Store the result
            MOVX @DPTR,A            ; in respective memory locations
            INC DPTR
            MOV A,0F0h
            MOVX @DPTR,A
    HERE:SJMP HERE                  ; End the program
```

Result: (8060)= 08h
(8061)= 02h

i.e., 65X8 = $(520)_{10}$ = 0208h

37. Write an 8051 program to divide the number in 8050h by the number in 8051h.Store the quotient and remainder in 8060h & 8061h of data memory respectively.

Algorithm:
1. Initialize the pointer DPTR & save the divisor in B reg
2. Again initialize the pointer DPTR & take the dividand in Acc register
3. Divide two 8-bit data
4. Store the result in the respective memory locations
4. End the program

```
Sample data: (8050) =41h     Dividend
             (8051)=08h      Divisor
```

Source Code:

```
        ORG 8000H               ; Starting address
        MOV DPTR,#8051h         ; Initialize DPTR
        MOVX A,@DPTR            ; Multiplier in A reg
        MOV 0F0h,A              ; Save Multiplier in B reg
        MOV DPTR,#8050h         ; Initialize DPTR
        MOVX A,@DPTR            ; Multiplicand in A reg
        DIV AB                  ; Multiply
        MOV DPTR,#8060h         ; Store the result
        MOVX @DPTR,A            ; in respective memory locations
        INC DPTR
        MOV A,0F0h
        MOVX @DPTR,A
HERE:SJMP HERE                  ; End the program
```

```
Result    : (8060) =08h     Quotient
            (8061)=01h      Reminder
```

Experiment No: 9
Programs using different addressing modes

38. Write an 8051 program to copy the value 55H into RAM memory locations 40H to 44H using

A) Direct addressing mode
B) Register addressing mode without using Loop and
C) With Loop

Algorithm:
1. Initialize the data in Acc register & copy it directly in the memory location for direct addressing mode
2. Initialize the data in Acc register which has to be moved
3. Move the data from Acc register to the pointer location
4. Repeat the process by incrementing the pointer for ‘n’ times (without a loop) & initialize the counter & repeat the loop foe ‘n’ times (with a loop)
5. End the program

Source Code:

A) Using direct addressing mode

```
          ORG 8000H        ; Starting address
          MOV A,#55h       ; Move the immediate data in A reg
          MOV 40h,A        ; Copy A to RAM Locations
          MOV 41h,A
          MOV 42h,A
          MOV 43h,A
          MOV 44h,A
     HERE:SJMP HERE        ; End the program
```

B) Using reg-indirect addressing mode without loop

```
            ORG 8000H          ; Starting address
            MOV A,#55h         ; Move the immediate data in A reg
            MOV R0,#40h        ; Initialize the pointer R0
            MOV @R0,A          ; Move data from A reg to R0 location
            INC R0             ; Increment the pointer
            MOV @R0,A
            INC R0
            MOV @R0,A
            INC R0
            MOV @R0,A
            INC R0
            MOV @R0,A
       HERE:SJMP HERE          ; End the program
```

C) With Loop

```
           ORG 8000H
            MOV A,#55h        ; Move the immediate data in A reg
            MOV R0,#40h       ; Initialize the pointer R0
            MOV R2,#05h       ; Initialize the counter
      AGAIN:MOV @R0,A         ; Move data from A reg to R0 location
            INC R0            ; Increment R0
            DJNZ R2,AGAIN ; Decrement & jump if R2!=0 to AGAIN
       HERE:SJMP HERE         ; End the program
```

39. Six bytes of data are stored in memory locations starting at 50H. Add all the bytes. Use register R7 to save any carries generated. Store the sum at memory locations 60H & 61H.

Algorithm:
1. Initialize the source pointer & counter registers
2. Clear Acc & the register to save carry
3. Add the data in Acc register with data in source pointer location
4. Check the carry flag i.e., if cy=1 then increment the register else repeat steps 3 & 4 till count becomes zero
5. Store the result in the desired memory locations
6. End the program

Sample data: (50)=10h, (51)=25h, (52)=2AH, (53)=4Fh, (54)=60h, (55)=3Fh

Source Code:

```
       ORG 8000H          ; Starting address
       MOV R0,#50h        ; Initialize pointer R0
       MOV R2,#06h        ; Initialize the counter R2
       CLR A              ; Initial sum=0
       MOV R7,A           ; Clear R7 to save carry
AGAIN:ADD A,@R0           ; Add data at R0 with A reg
       JNC NEXT           ; Jump if cy=0 to NEXT
       INC R7             ; Keep track of carries
NEXT: INC R0              ; Increment pointer
       DJNZ R2,AGAIN      ; Decrement & jump if R2!=0 to AGAIN
       MOV 60h,A          ; Store LSBy of sum
       MOV 61h,R7         ; Store MSBy of sum
 HERE:JMP HERE            ; End the program
```

Result = (60)=4Dh
(61)=01h (MS byte) ; 014Dh

Format: For entering, executing & Checking results.

Enter source code

<EXM MEM> <PRG MEM> 8000 <NXT> DATA <NXT>……. <EXEC>

Feed Sample Data

<EXM MEM> <INTDATA> 50 <NXT> DATA <NXT> DATA…….. <EXEC>

<EXM MEM> <INTDATA> 60 <NXT>00XT>00<EXEC>

Run the Program

<GO><8000> <EXEC>

Reset

Check Results

<EXM MEM> <INT DATA> 60 <NXT>….

40. Write an 8051 program to copy a block of 10 bytes of data from RAM locations starting at 35h to RAM locations starting at 60h.

Algorithm:
1. Initialize the source pointer, destination pointer & a counter register
2. Move the data from source pointer register to the Acc register
3. Move the data from Acc register to the destination pointer location
4. Increment both the pointer registers
5. Repeat the loop for ‘n’ times
6. End the program

Sample Prob
Source block

(35)=10h, (36) =20h, (37) = 30h, (38) = 40h, (39) = 50h,
(3A) = 60h, (3B) = 70h, (3C) = 80h, (3D) = 90h, (3E) =A0h

Source Code:

```
               ORG 8000H              ; Starting address
               MOV R0,#35h            ; Source pointer
               MOV R1,#60h            ; Destination pointer
               MOV R3,#0Ah            ; Counter
          BACK:MOV A,@R0              ; Move data from R0 to A reg
               MOV @R1,A              ; Move data from A reg to R0 location
               INC R0                 ; Increment R0
               INC R1                 ; Increment R1
               DJNZ R3,BACK           ; Decrement & jump if R3!=0 to BACK
          HERE:SJMP HERE              ; End the program
```

Result: (60)=10h, (61) =20h, (62) = 30h, (63) = 40h, (64) = 50h,
(65) = 60h, (66) = 70h, (67) = 80h, (68) = 90h, (69) =A0h

41. A byte is stored in register R0. Write an 8051 Program to find the number of 1's in a byte stored in R0 and Store the number of 1's in register R2.

Algorithm:
1. Initialize Acc register with the byte for which number of ones has to be counted & counter register
2. Rotate the byte one bit left with carry & check the carry flag
3. If cy=1 then increment result register
else decrement the count by 1 & repeat the steps 2 & 3 till the count becomes zero
4. End the program

Source Code:

```
        Let R0 = AAh    i.e.,  10101010

        ORG 8000H            ; Starting address
        MOV R0,#AAh          ; Move data in R0
        MOV A,R0             ; Take it in A reg
        MOV R2,#00h          ; Clear R2 reg
        MOV R1,#08h          ; Initialize the counter R1
LOOP:RLC A                   ; Rotate left with  cy
        JNC CONT             ; Jump if cy=0 to CONT
        INC R2               ; Increment R2
CONT:DJNZ R1,LOOP            ; Decrement & jump if R1!=0 to LOOP
HERE:SJMP HERE               ; End the program
```

Result: R2=04h

42. Write an 8051 program to find the number 64h from the set of five readings starting from address location 50H to 54h. If present store 00h in R0, otherwise store FFh in R0.

Algorithm:
1. Initialize the source pointer & counter register
2. Compare the byte in source pointer location with the byte which has to be searched
3. If both the bytes are equal, store 00h in desired register else repeat steps 2 & 3 till the count becomes zero
4. If the byte is not found, store FFh in desired register
5. End the program

Sample Problem (1):

(50) =76h, (51) =45h, (52) =64h, (53) =25h, (54) =22h

Source Code:

```
              ORG 8000H               ; Starting address
              MOV R1,#50h             ; Initialize the source pointer R1
              MOV R2,#05h             ; Initialize the counter R2
       LOOP:CJNE @R1,#64H,CONT        ; Compare & jump not equal to CONT
              MOV R0,#00h             ; Store 00H in R0 reg
      HERE1:SJMP HERE1                ; End the program
       CONT:INC R1                    ; Increment R1 reg
             DJNZ R2,LOOP             ; Decrement & jump if R2!=0 to LOOP
             MOV R0,#FFh              ; Store FFH in R0 reg
     HERE2:SJMP HERE2                 ; End the program
```

Result = (R0) = 00h

Sample prob (2)

Replace data in (52) by 94h
Result = (R0) = FFh

Experiment No: 10
Interfacing 8051 with DAC to generate waveforms

AIM: To write and execute program in 8051 assembly language for interfacing a DAC interface module with ESA 31 microcontroller trainer kit.

APPARATUS:

1. ESA 31 Microcontroller trainer kit
2. Dual channel DAC module
3. Power supply units
4. 26 Pin connector cable
5. CRO

DESCRIPTION:

To use DAC, initialize 8255A for mode 0 operation with port A and port B as output. Output data on the appropriate port and observe output wave form at Xout and Yout of the DAC using CRO.

The 16 bit port addresses for 8255A available at J2 connector are:

Port A	Equ	E800H
Port B	Equ	E801H
Port C	Equ	E802H
Port D	Equ	E803H

Note: Port A controls Xout and Port B controls Yout of DAC interface module.

PROGRAMS:

- Write an ALP to generate Saw tooth (Up-going and Down-going)
- Write an ALP to generate Triangular waveform
- Write an ALP to generate Symmetrical Square wave
- Write an ALP to generate
    - ✓ Up-going stair case with 5 steps
    - ✓ Down-going stair case with 5 steps

; Assume the DAC interface is connected over J2 of the ESA 31 trainer.

```
ORG                 8000H
PORT_A     EQU      E800H
PORT_B     EQU      E801H
PORT_C     EQU      E802H
CWR        EQU      E803H
```

43. Program to generate Continuous up going saw tooth.

Algorithm:
1. Initialize 8255 in mode 0 & all ports as output ports
2. Initialize A with 00h
3. Send the data to the ports A & B through A register
4. Increment A
5. Repeat the steps 3 & 4 for the generation of continuous waveform

Source Code:

```
          ORG 8000H              ; Starting address
          MOV DPTR,#0E803H
          MOV A,#80H             ; Initialize 8255A for mode 0
          MOVX @DPTR,A           ; with $P_A$ & $P_B$ as OUT
          CLR A                  ; Start with value 00H
    AGAIN:MOV DPTR, #0E800H      ; Point to Port A
          MOVX @DPTR,A           ; Out to Port A
          INC DPTR               ; Increment DPTR
          MOVX @DPTR,A           ; Out to Port B
          INC A                  ; Increment DAC input
          SJMP   AGAIN           ; Repeat forever
```

Result: The output waveform is observed on CRO & amplitude and time period is measured.

44. Program to generate continuous down going saw tooth

Algorithm:

1. Initialize 8255 in mode 0 & all ports as output ports
2. Initialize A with FFh
3. Send the data to the ports A & B through A register
4. Decrement A
5. Repeat the steps 3 & 4 for the generation of continuous waveform

Source Code:

```
             ORG 8000H                  ; Starting address
             MOV DPTR,#0E803H
             MOV A,#80H                 ; Initialize 8255A for mode 0
             MOVX @DPTR,A               ; with $P_A$ & $P_B$ as OUT
             MOV A,#0FFH                ; Start with value FFH
       AGAIN:MOV DPTR,#0E800H           ; Point to Port A
             MOVX @DPTR,A               ; Out to Port A
             INC DPTR                   ; Increment DPTR
             MOVX @DPTR,A               ; Out to Port B
             DEC A                      ; Decrement DAC input
             SJMP AGAIN                 ; Repeat forever
```

Result: The output waveform is observed on CRO & amplitude and time period is measured.

45. Program to generate continuous triangular waveform

Algorithm:
1. Initialize 8255 in mode 0 & all ports as output ports
2. Initialize A with 00h
3. Send the data to the ports A & B through A register
4. Increment A
5. Repeat the steps 3 & 4 for the generation of up-going saw-tooth waveform
6. Repeat the steps 2-5 with A = FFh & decrementing A for the generation of down-going saw-tooth
7. Repeat the steps 2-6 for the generation of continuous waveform

Source Code:

```
          ORG 8000H            ; Starting address
          MOV DPTR,#0E803H
          MOV A,#80H           ; Initialize 8255A for mode 0
          MOVX @DPTR,A         ; with PA & PB as OUT
          CLR A                ; Start with value 00H
       UP:MOV DPTR,#0E800H     ; Point to Port A
          MOVX @DPTR,A         ; Out to Port A
          INC DPTR             ; Increment DPTR
          MOVX @DPTR,A         ; Out to Port B
          INC A                ; Increment DAC input
          CJNE A,#0FFH,UP      ; Compare & jump if  A!=FFH to UP
     DOWN:MOV DPTR,#0E800H     ; Point to Port A
          MOVX @DPTR,A         ; Out to Port A
          INC DPTR             ; Increment DPTR
          MOVX @DPTR,A         ; Out to Port B
          DEC A                ; Decrement DAC input
          CJNE A,#00H,DOWN     ; Compare & jump if  A!=00H to DOWN
          SJMP  UP             ; Repeat forever
```

Result: The output waveform is observed on CRO & amplitude and time period is measured.

46. Program to generate Symmetrical Square Wave

Algorithm:
1. Initialize 8255 in mode 0 & all ports as output ports
2. Initialize A with 00h
3. Send the data to the port A through A register & provide the delay
4. Make A = FFh, send it through port A & provide the delay
5. Repeat the steps 2-4 for the generation of continuous waveform

Source Code:

```
       ORG 8000H              ; Starting address
       MOV DPTR,#0E803H
       MOV A,#80H             ; Initialize 8255A for mode 0
       MOVX @DPTR,A           ; with $P_A$ & $P_B$ as OUT
BACK:  MOV A,#0FFH            ; Start with value FFH
       MOV DPTR,#0E800H       ; Point to Port A
       MOVX @DPTR,A           ; Out to Port A
       MOV R0,#0FFH           ; Move FFH to R0
DLY1:  DJNZ R0,DLY1           ; for delay
       MOV A,#00H             ; Now start with value FFH
       MOVX @DPTR,A           ; Out to Port A
       MOV R0,#0FFH           ; Move FFH to R0
DLY2:  DJNZ R0,DLY2           ; for same delay
       SJMP BACK              ; Repeat forever
```

Result: The output waveform is observed on CRO & amplitude and time period is measured.

47. Program for Stair case (Up-going) with 5 steps

Algorithm:
1. Initialize 8255 in mode 0 & all ports as output ports
2. Initialize A with 00h
3. Add 33h to A & send the data to the port A through A register & provide the delay
4. Compare A with FFh & Repeat the steps 2 & 3 for the generation of continuous waveform

Source Code:

```
          ORG 8000H              ; Starting address
          MOV DPTR,#0E803H
          MOV A,#80H             ; Initialize 8255A for mode 0
          MOVX @DPTR,A           ; with PA & PB as OUT
          MOV A,#00H             ; Start with value 00H
      RPT:ADD A,#33H             ; Add 33H to A reg
          MOV DPTR,#0E800H       ; Point to Port A
          MOVX @DPTR,A           ; Out to Port A
          MOV R0,#0FFH           ; Move FFH to R0
     DLY1:DJNZ R0,DLY1           ; for delay
          CJNE A,#0FFH,RPT       ; Compare & jump if A!=FFH to RPT
          INC A                  ; Increment A reg
          MOVX @DPTR,A           ; Out to Port A
          MOV R0,#0FFH           ; Move FFH to R0
     DLY2:DJNZ R0,DLY2           ; for delay
          SJMP RPT               ; Repeat forever
```

Result: The output waveform is observed on CRO & amplitude and time period is measured.

48. Program for Stair case (down-going) with 5 steps

Algorithm:

1. Initialize 8255 in mode 0 & all ports as output ports
2. Initialize A with FFh
3. Subtract 33h from A & send the data to the port A through A register & provide the delay
4. Compare A with 00h & Repeat the steps 2 & 3 for the generation of continuous waveform

Source Code:

```
          ORG 8000H              ; Starting address
          MOV DPTR,#0E803H
          MOV A,#80H             ; Initialize 8255A for mode 0
          MOVX @DPTR,A           ; with $P_A$ & $P_B$ as OUT
          MOV A,#0FFH            ; Start with value FFH
          MOV DPTR,#0E800H       ; Point to Port A
      RPT:MOVX @DPTR,A           ; Out to Port A
          MOV R0,#0FFH           ; Move FFH to R0
     DLY1:DJNZ R0,DLY1           ; for delay
    AGAIN:CLR C                  ; Clear cy
          SUBB A,#33H            ; Subtract 33H from A reg
          MOVX @DPTR,A           ; Out to Port A
          MOV R0,#0FFH           ; Move FFH to R0
     DLY2:DJNZ R0,DLY2           ; for delay
          CJNE A,#00H,AGAIN      ; Compare & jump if A!=00H to AGAIN
          DEC A                  ; Decrement A reg
          SJMP RPT               ; Repeat forever
```

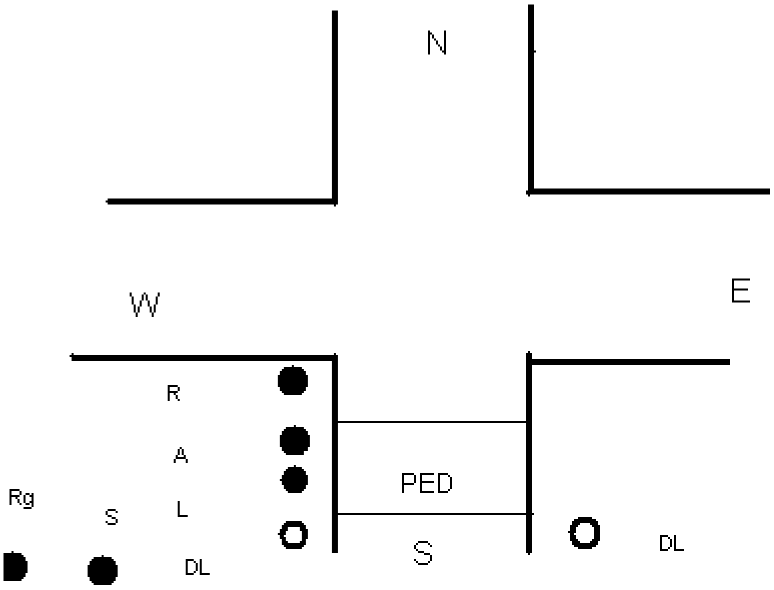
Result: The output waveform is observed on CRO & amplitude and time period is measured.

EXPERIMENT NO.11
Interfacing of Traffic Light Controller Using 8051

The traffic light interface simulates the control and operation of traffic lights at a junction of four roads. The interface provides a set of 6 LED indicators at each of the four corners. Each of these LED s can be controlled by a port line. Thus the interface allows the user to simulate a variety of traffic simulations using appropriate software routines.

DESCRIPTION OF THE CIRCUIT:

The organization of 6 LED s is identical at each of the four corners. The organization with reference to the LED s at "South-West" corner is shown in the figure below:



R = SOUTH RED
A = SOUTH AMBER
L = SOUTH LEFT
S= SOUTH STRAIGHT
Rg=SOUTH RIGHT
DL=SOUTH PEDESTRIAN

The five LED s (except "Pedestrian") will be ON or OFF depending on the state of corresponding port line LED is ON, if the Port line is Logic 'HIGH' and LED is OFF, if it is at logic 'LOW'. The last LED marked DL is a set of two dual color LED s and they both will be either RED or GREEN depending on the state of the corresponding port line RED if the port line is logic HIGH and GREEN if the port line is logic LOW.

24 LEDS AND CORRESPONDING PORT LINES:

PORT A:

D7	D6	D5	D4	D3	D2	D1	D0
ER	EA	ERg	EL	SR	SA	SRg	SL

PORT B:

D7	D6	D5	D4	D3	D2	D1	D0
WR	WA	WRg	WL	NR	NA	NRg	NL

PORT C:

D7	D6	D5	D4	D3	D2	D1	D0
EP	SP	WP	NP	SS	ES	NS	WS

There are four such sets of LED s and these are controlled by 24 port lines of 8255A. Each port line is inverted and buffered using 7406 (open collector inverter buffers) and is used to control an LED. Dual color LEDs are controlled by a port line and its complement.

INSTALLATION:

The interface module has 26-pin connector at one edge of the card. This is used for connecting the interface over J2 of the ESA 31 trainer. The trainer can be in KEYBOARD MODE or SERIAL MODE.

49. PROBLEM STATEMENT:

Generate the sequence for P_A, P_B, and P_C such that the following traffic situations are simulated.

1. Vehicles from SOUTH can go NORTH and WEST
   Vehicles from WEST can go NORTH
   Vehicles from NORTH can go SOUTH
   Pedestrians can cross on EAST

2. Vehicles from EAST can go WEST and SOUTH
   Vehicles from WEST can go EAST
   Vehicles from SOUTH can go WEST
   Pedestrians can cross on NORTH

3. Vehicles from EAST can go SOUTH
   Vehicles from NORTH can go SOUTH and EAST
   Vehicles from SOUTH can go NORTH
   Pedestrians can cross on WEST

4. Vehicles from EAST can go WEST
   Vehicles from WEST can go EAST and NORTH
   Vehicles from NORTH can go EAST
   Pedestrians can cross on SOUTH

5. No vehicle movement
   Pedestrians can cross on all four roads.

The system moves from one state to another state after fixed time delay. The state transition is indicated by turning ON all the AMBER LEDs and all Pedestrians RED LEDs for a fixed duration. The sequence of the above states is repeated again and again.

Algorithm:
1. Initialize 8255 in mode 0 & all ports as output ports
2. Load the look-up table with port values according to the traffic situations
3. Send the port values through the respective port addresses
4. Provide the delay in between the two states
5. Repeat the process to control the traffic continuously

```
; Program memory from 8000H to 804FH
                ORG                         8000H
                PORT A          EQU         E800H
                PORT B          EQU         E801H
                PORT C          EQEU        E802H
                CWR             EQU         E803H
```

```
          MOV DPTR,#0E803H
          MOV A,#80H                 ; Initialize 8255A for mode 0
          MOVX @DPTR,A               ; with $P_A$ & $P_B$ as OUT
    AGAIN:MOV DPTR,#PORTS            ; Initialize the PORTS address
   NEXTST:MOVX A,@DPTR
          PUSH DPL
          PUSH DPH
          MOV DPTR,#0E800H         ;Port A value
          MOVX @DPTR,A
          POP DPH
          POP DPL
          INC DPTR
          MOVX A,@DPTR
          PUSH   DPL
          PUSH DPH
          MOV DPTR,0E801H         ;Port B value
          MOVX @DPTR,A
          POP DPH
          POP DPL
          INC DPTR
          MOVX A,@DPTR
          PUSH   DPL
          PUSH   DPH
          MOV DPTR,#0E802H      ; Port C value
          MOVX @DPTR,A
          POP DPH
          POP DPL
          INC DPTR
          LCALL DELAY               ; Provide delay
          MOV A,DPL
          CJNE A,#1EH,NEXTST
          SJMP AGAIN
```

```
DELAY:MOV R2,#06                ; Delay routine
 LOOP3:MOV R4,#0FFH
 LOOP2:MOV R3, #0FFH
 LOOP1:DEC R3
       CJNE R3,#00H,LOOP1
       DEC R4
       CJNE R4,#00H,LOOP2
       DEC R2
       CJNE R2,#00H,LOOP3
       RET
```

; Enter the data mentioned below from 0000H to 001EH in data memory.

```
PORTS:   DB   10H, 81H, 7AH      ;State 1
         DB   44H, 44H, 0F0H     ;All Ambers ON
         DB   08H, 11H, 0E5H     ;State 2
         DB   44H, 44H, 0F0H     ;All Ambers ON
         DB   81H, 10H, 0DAH     ;State 3
         DB   44H, 44H, 0F0H
         DB   11H, 08H, 0B5H     ;State 4
         DB   44H, 44H, 0F0H
         DB   88H, 88H, 00H      ;State 5
         DB   44H, 44H, 0F0
         DB   00H                ;Dummy
```

Result: The output is observed on traffic light interface module.

50. The following sequence of simple traffic conditions are simulated as:

Condition 1

- Vehicles from SOUTH can go NORTH and WEST
- Vehicles from WEST can go NORTH
- Vehicles from NORTH can go SOUTH
- Pedestrian can cross on EAST

Condition 2

- No vehicle movement
- Pedestrians can cross on all four roads

Algorithm:
1. Initialize 8255 in mode 0 & all ports as output ports
2. Load the look-up table with port values according to the traffic situations
3. Send the port values through the respective port addresses
4. Provide the delay in between the two states
5. Repeat the process to control the traffic continuously

```
PORTS: Db A0h, 81h, 7Ah
       Db 44h, 44h, 0F0h
       Db 88h, 88h, 00h
       Db 44h, 44h, 0F0h
     Db 00h
```

EXPERIMENT NO.12
Programs to control stepper motor using 8051

Data acquisition and control represents the most popular applications of microprocessors & microcontrollers. Stepper motor control is a very popular application of microprocessors & microcontrollers in control area as stepper motors are capable of accepting pulses directly from the processors & controllers and move accordingly.

There are two types of Stepper motors:

1. Permanent Magnet (PM)
2. Variable Reluctance (VR)

OPERATION OF STEPPER MOTOR:

Stepper motor consists of two important parts, the stator and the rotor. The stator normally has 4 windings on four wheels whereas the rotor is magnetic in nature and has got teeth on it, which is magnetized as North and South poles.

WORKING:

Stepper motor works on the principle of repulsion between magnets. One input to the stepper motor is given in the form of pulses, provided to the windings on the poles as 1000, 0100, 0010, 0001. The windings are provided with input by the 8051 microcontroller through the Port A pins of 8255.

Stator is responsible for creating the magnetic field and rotating the rotor.

SPECIFICATIONS OF THE STEPPER MOTOR USED:

The motor is reversible on the application of a torque of 3Kgcm. The power requirement is +5V DC at 1.2A current per winding at full torque. The step angle is 1.8°, i.e., for every single excitation, the motor shaft rotates by 1.8°.For the motor to rotate one full revolution (360°), number of steps required is

$$360^{o} / 1.8^{o} = 200$$

The stepper motor used has four stator windings which are brought out through colored wires terminated at a 4 pin polarized female connector. The remaining two wires (White & Black wires) are shorted and terminated at 2 pin polarized female connector.

LOOPING:

The number of times the stepper motor should loop is given by:

Count = No. of teeth on rotor X total No. of rotations.

The Port A pins of 8255 (PA0, PA1, PA2, PA3) are used. The values that have to be sent to Port A to drive the stepper motor in clock wise direction are 88h, 44h, 22h, 22h and anti clock wise direction are 11h, 22h, 44h, 88h.

CIRCUIT DESCRIPTION:

The stepper motor interface uses four transistor pairs (SL100 & 2N3055) in a Darlington pair configuration. Each Darlington pair is used to excite the particular winding of the motor connected to 4 pin connector on the interface. The inputs to these transistors are from the 8255 PPI I/O lines of the microcontroller trainer kit. ‘Port A’ lower nibble PA0, PA1, PA2, PA3 is the four lines brought out to the 26 pin FRC male connector J1 on the interface module. The freewheeling diodes across each winding protect transistors from switching transients.

INSTALLATION:

The interface has two no. of 3 pins and one four pin connectors. Plug in four pin polarized connector of the motor to interface and the 3 pin connector of the motor to the 3 pin connector of the interface marked as “WHT BLK”. Connect the 3 pin female connector of the stepper motor power supply to the connector on the interface marked as “GND +5V/12V”. Connect the 26 core flat ribbon cable to J1 connector on the interface module and the other end of the cable to microcontroller 8051 trainer kit J2.

Switch on power to the trainer kit as well as the stepper motor. Key in the program required for the application and executes the same. When the program is executed, the motor shaft rotates in steps at the speed depending upon the delay between successive steps, which is generated and can be controlled by the program. The direction of rotation can also be controlled through software.

CALCULATIONS:

No. of teeth on rotor = N1 = 50
No. of poles on stator = 8
No. of teeth on stator = 8X5 = N2 = 40

$$\text{Step angle} = \frac{360^{o}\,(N1 - N2)}{N1 * N2} = 1.8^{O}$$

The step angle is 1.8^{O} i.e. for every single excitation; the motor shaft rotates by 1.8^{O}.

51. Write an 8051 program to drive the Stepper motor continuously in clockwise direction.

Algorithm:
1. Initialize 8255 in mode 0 & all ports as output ports
2. Load the input pattern in Acc register & send it through port A address
3. Rotate right the value of Acc register
4. Provide the delay
5. Repeat the process for continuous rotation

Source Code:

```
      ORG 8000H               ; Starting address
      MOV DPTR, #0E803H
      MOV A, #80H             ; Initialize 8255A for mode 0
      MOVX @DPTR, A           ; with PA & PB as OUT
      MOV A, #88H             ; Move the input pattern to A
BACK:MOV DPTR, #0E800H        ; Point to Port A
      MOVX @DPTR, A           ; Out to Port A
      RR A                    ; Rotate right A reg
      MOV R4, #10H            ; Delay routine
LOOP:MOV R3, #0FFH
DLY1:DJNZ R3, DLY1
      DJNZ R4, LOOP
      SJMP BACK              ; Repeat continuously
```

Result: The motor shaft rotates continuously in clockwise direction.

52. Write an 8051 program to drive the Stepper motor continuously in anti-clockwise direction.

Algorithm:
1. Initialize 8255 in mode 0 & all ports as output ports
2. Load the input pattern in Acc register & send it through port A address
3. Rotate left the value of Acc register
4. Provide the delay
5. Repeat the process for continuous rotation

Source Code:

```
            ORG 8000H               ; Starting address
            MOV DPTR, #0E803H
            MOV A, #80H             ; Initialize 8255A for mode 0
            MOVX @DPTR, A           ; with PA & PB as OUT
            MOV A, #88H             ; Move the input pattern to A
BACK:MOV DPTR, #0E800H              ; Point to Port A
            MOVX @DPTR, A           ; Out to Port A
            RL A                    ; Rotate left A reg
            MOV R4, #10H            ; Delay routine
LOOP:MOV R3, #0FFH
DLY1:DJNZ R3, DLY1
            DJNZ R4, LOOP
            SJMP BACK               ; Repeat continuously
```

Result: The motor shaft rotates continuously in anti-clockwise direction.

53. Write an 8051 program to drive the Stepper motor 5 times clockwise & 3 times anti-clockwise direction.

Algorithm:
1. Initialize 8255 in mode 0 & all ports as output ports
2. Initialize the counter for clockwise rotation & register for step size
3. Load the input pattern in Acc register & send it through port A address
3. Rotate right the value of Acc register
4. Provide the delay
5. Repeat the process for clockwise rotation till the count becomes zero
6. Repeat the process for anti-clockwise rotation
7. Repeat the steps 2-6 for continuous rotation

Source Code:

```
              ORG 8000H              ; Starting address
              MOV DPTR,#0E803H
              MOV A,#80H             ; Initialize 8255A for mode 0
              MOVX @DPTR,A           ; with PA & PB as OUT
              MOV R1,#05H            ; Initialize the counter
              MOV R0,#0C8H           ; Initialize the step size
              MOV A,#88H             ; Move the input pattern to A
        BACK1:MOV DPTR,#0E800H       ; Point to Port A
              MOVX @DPTR,A           ; Out to Port A
              RR A                   ; Rotate right A reg
              MOV R4,#10H            ; Delay routine
        LOOP1:MOV R3,#0FFH
         DLY1:DJNZ R3,DLY1
             DJNZ R4,LOOP1
             DJNZ R0,BACK1
             DJNZ R1,BACK1
              MOV R1,#03H            ; Initialize the counter
              MOV R0,#0C8H           ; Initialize the step size
              MOV A,#11H             ; Move the input pattern to A
        BACK2:MOVX @DPTR,A           ; Out to Port A
```

```
              RL A                    ; Rotate left A reg
              MOV R4,#10H             ; Delay routine
       LOOP2:MOV R3,#0FFH
        DLY2:DJNZ R3,DLY2
              DJNZ R4,LOOP2
              DJNZ R0,BACK2
             DJNZ R1,BACK2
             SJMP BACK1             ; Repeat continuously
```

Result: The motor shaft rotates in clockwise direction 5 times & anti-clockwise direction 3 times.

APPENDIX

LABORATORY COURSE ASSESSMENT GUIDELINES

i. The number of experiments/programs/sessions in each laboratory course shall be as per the curriculum in the scheme of instructions provided by OU.

ii. The students will maintain a separate note book for each laboratory course in which all the related work would be done.

iii. In each session the students will complete the assigned tasks of process development, coding, compiling, debugging, linking and executing the programs.

iv. The students will then execute the programme and validate it by obtaining the correct output for the provided input. The course coordinator will certify the validation in the same session.

v. The students will submit the record in the next class. The evaluation will be continuous and not cycle-wise or at semester end.

vi. The internal marks of 25 are awarded in the following manner:

a.	Laboratory record	-	Maximum Marks 15
b.	Test and Viva Voce	-	Maximum Marks 10

vii. Laboratory Record: Each experimental record is evaluated for a score of 50. **The rubric parameters are as follows:**

a.	Write up format	-	Maximum Score 20
b.	Process development and coding	-	Maximum Score 10
c.	Compile, debug, link and execute program	-	Maximum Score 15
d.	Process validation through input-output	-	Maximum Score 5

While (a) is assessed at the time of record submission, (b), (c) and (d) are assessed during the session based on the performance of the student in the laboratory session. Hence if a student is absent for any laboratory session but completes the program in another session and subsequently submits the record, it shall be evaluated for a score of 20 and not 50.

viii. The experiment evaluation rubric is therefore as follows :

ix.

Parameter	Max Score	Outstanding	Accomplished	Developing	Beginner	Points
Process Development and Coding	10					
Compilation, Debugging, Linking and Executing	15					
Process Validation	5					
Write up format	20					

x. The first page of the record will contain the following title sheet:

MUFFAKHAM JAH COLLEGE OF ENGINEERING AND TECHNOLOGHY

LABORATORY EXPERIMENT ASSESSMENT SHEET

ELECTRONICS & COMMUNICATION ENGINEERING DEPARTMENT

B.E. 3/4 2018-2019

MICROPROCESSOR & MICROCONTROLLER LABORATORY

NAME: **ROLL NO.**

Exp. No.	Title of the Program	Date conducted	Date Submitted	Process Development and Coding (Max 10)	Compilation , Debugging, Linking and Executing (Max 15)	Process Validation (Max 5)	Write up format (Max 20)	Total Score (Max 50)
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								
11								
12								
	TOTAL							

Date: **Signature of Course Coordinator**

xi. The 15 marks of laboratory record will be scaled down from the TOTAL of the assessment sheet.

xii. The test and viva voce will be scored for 10 marks as follows:

Internal Test	-	6 marks
Viva Voce / Quiz	-	4 marks

xiii. Each laboratory course shall have 5 course outcomes.

The proposed course outcomes would be as follows:

On successful completion of the course, the student will acquire the ability to:

1. Apply the design concepts for development of a process and interpret data
2. Demonstrate knowledge of programming environment, compiling, debugging, linking and executing variety of programs.
3. Demonstrate documentation and presentation of the algorithms / flowcharts / programs in a record form.
4. Validate the process using known input-output parameters.
5. Employ analytical and logical skills to solve real world problem and demonstrate oral communication skills.

xiv. The Course coordinators would prepare the assessment matrix in accordance with the guidelines provided above for the five course outcomes. The scores to be entered against each of the course outcome would be the sum of the following as obtained from the assessment sheet in the record:

a. Course Outcome 1: Sum of the scores under ‘Process Development and Coding’.
b. Course Outcome 2: Sum of the scores under ‘Compilation/Debugging/Linking and Executing’.
c. Course Outcome 3: Sum of the scores under ‘Write up format’.
d. Course Outcome 4: Sum of the scores under ‘Process validation’.
e. Course Outcome 5: Marks for ‘Internal Test and Viva voce’.

xv. Soft copy of the assessment matrix would be provided to the course coordinators.

xvi. There may be some laboratory courses based on proprietary software like MATLAB, AUTOCAD etc. for which the course coordinators and programme coordinators would formulate appropriate course outcomes.

MUFFAKHAM JAH COLLEGE OF ENGINEERING AND TECHNOLOGY

Program Outcomes of B.E (ECE) Program:

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, research literature, and analyse 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.

PO 12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes (PSOs) of ECE Department, MJCET

PSO1: The ECE Graduates will acquire state of art analysis and design skills in the areas of digital and analog VLSI Design using modern CAD tools.

PSO2: The ECE Graduates will develop preliminary skills and capabilities necessary for embedded system design and demonstrate understanding of its societal impact.

PSO3: The ECE Graduates will obtain the knowledge of the working principles of modern communication systems and be able to develop simulation models of components of a communication system.

PSO4: The ECE Graduates will develop soft skills, aptitude and programming skills to be employable in IT sector.