UNIT –II

(ASSEMBLY LANGUAGE PROGRAMMING)

Syllabus: Assembly language programs involving logical, branch and call instructions, sorting, evaluation of arithmetic expressions, string manipulation.

INTRODUCTION TO PROGRAMMING THE 8086

Programming Languages: To run a program, a microcomputer must have the program stored in binary form in successive memory locations. There are three language levels that can be used to write a program for a microcomputer.

1. Machine Language
2. Assembly Language
3. High-level Languages

Machine Language: You can write programs as simply a sequence of the binary codes for the instructions you want the microcomputer to execute. This binary form of the program is referred to as *machine language* because it is the form required by the machine. However, it is very difficult, not possible, for a programmer to memorize the thousands of binary instruction codes for a microprocessor**.** Also, it is very easy for an error to occur when working with long series of 1's and 0's. Using hexadecimal representation for the binary codes might help some, but there are still thousands of instruction codes to cope with.

Assembly Language: To make programming easier, many programmers write programs in *assembly language.* They then translate the assembly language program to machine language so that it can be loaded into memory and run. Assembly language uses 2, 3, or 4- letter *mnemonics* to represent each instruction type. A mnemonic is advice to help you remember something. The letters in an assembly language mnemonic are usually initials or shortened form of the English word(s) for the operation performed by the instruction. For example, the mnemonic for addition is ADD, the mnemonic for subtraction is SUB and the mnemonic for the instruction to copy data from one location to another is MOV. Assembly language statements are usually written in a standard form that has *four fields,* as shown in fig. below.

LABEL FIELD	OPCODE/MNEMONIC FIELD	OPERAND FIELD	COMMENT FIELD
NEXT:	ADD	AL,07H	;Add immediate number 07H to the contents of AL register

Fig. Assembly Language statement format.

The first field in an assembly language statement is the *Label field.* A label is a symbol or group of symbols used to represent an address which is not specially known at the time the statement is written. Labels are usually followed by a colon.

The *opcode field* of the instruction contains the mnemonic for the instruction to be performed. Instruction mnemonics are sometimes called *operation codes* or *opcodes.*

The *operand field* of the statement contains the data, the memory address. The port address, or the name of the register on which the instruction is to be performed. Operand is just another name for the data item(s) acted on by the instruction. In the above example there are two operands, AL and 07H, specified in the operand field. AL represents the AL register, and 07H represents the number 07H. This assembly language statement thus says, "*Add the number 07H to the contents of the AL register."* By Intel convention, the result of the addition will be put in the register or the memory location specified before the comma in the operand field. For the example, the result will be left in the register AL.

The final field in an assembly language statement is *comment field,* which starts with a semicolon. Comments do not become the part of the machine language program, but they are very important.

High-level Language: Another way of writing a program for a microcomputer is with a *high-level language,* such as BASIC, Pascal, or C. These language use program statements which are even more English-like than those of assembly language. Each high level statement may represent many machine code instructions. An interpreter or a compiler program is used to translate higher-level language statements to machine codes. Programs can usually be written faster in high level languages than in assembly language because a high –level language work with bigger building blocks. However, programs written in a high –level language and interpreted or compiled almost always execute more slowly and require more memory than the same program written in assembly language.

Programs that involve a lot of hardware control, such as robots and factory control systems, or programs that must run as quickly as possible are usually best written assembly language. Complex data processing programs that manipulate massive amounts of data, such as insurance company records, are usually best written in a high-level language.

PROGRAM DEVELOPMENT STEPS

Developing a program however requires more than just writing down series of instructions. When you write a computer program, it is good idea to start by developing a detailed plan or outline for the entire program. You should ***never*** start writing an assembly language program by just writing down instructions!

The program development steps are:

1. Defining a Problem
2. Representing program operations
3. Finding the right instruction
4. Writing a program

ASSEMBLY LANGUAGE PROGRAM DEVELOPMENT TOOLS

For all but the very simplest assembly language programs, you will probably want to use some type of microcomputer development system and *program development tools* to make your work easier. Most of the program development tools are programs which you run to perform some function on the program you are writing.
Program development tools are:

1. Editor
2. Assembler
3. Linker
4. Locator
5. Debugger
6. Emulator

Editor: An editor is a program which allows you to create a file containing the assembly language statements for your program. When you have typed in your entire program, you then save the file on a hard disk. This file is called source file. The next step is to process the source file with an assembler. If you are going to use the TASM or MASM assembler, you should give your source file name the extension .ASM.

Assembler: An assembler is programming tool which is used to translate the assembly language mnemonics for instructions to the corresponding binary codes. The assembler generates two files. The first file, called the *object file*, is given the extension .OBJ. The object file contains the binary codes for the instructions and information about the addresses of the instructions. After further processing the contents of this file will be loaded into memory and run. The second file generated by the assembler is called the *assembler list file* and is given the extension .LST.

Linker: The linker is program used to join several object files into one large object file. The linkers which come with the TASM or MASM assemblers produce link files with the .EXE extension.

Locator: A locator is a program used to assign the specific addresses of where the segments of object code are to be loaded into memory.

Debugger: If your program requires no external hardware or requires only hardware accessible directly from your microcomputer, then you can use debugger to run and debug your program. A debugger is a program which allows you to load your object code program into system memory, execute the program, and troubleshoot or' debug' it.

Emulator: Another way to run your program is with an emulator. An emulator is a mixture of hardware and software. It is usually used to test and debug the hardware and software of an external system.

- **ASSEMBLY LANGUAGE PROGRAMS**

Simple programs

1. Write an ALP in 8086 to perform an addition of two 8-bit numbers.

```
        ASSUME CS: CODE
        ORG 2000H
CODE    SEGMENT
START:  MOV SI, 3000H
        MOV AL, [SI]
        INC SI
        MOV BL, [SI]
        ADD AL, BL
        INT 03H
CODE    ENDS
        END
```

Using data segment declaration

```
        ASSUME CS: CODE, DS: DATA
DATA    SEGMENT
        N1      DB      08H
        N2      DB      02H
DATA    ENDS
        ORG 3000H
CODE    SEGMENT
        MOV AX, DATA
        MOV DS, AX
        MOV AL, N1
        MOV BL, N2
        ADD AL, BL
        INT 03H
CODE    ENDS
        END
```

2. Write an ALP in 8086 to perform subtraction of two 8-bit numbers.

```
ASSUME CS: CODE
        ORG 2000H
CODE    SEGMENT
```

```
        MOV SI, 3000H
        MOV AL, [SI]
        INC SI
        MOV BL, [SI]
        SUB AL, BL
        INT 03H
CODE    ENDS
        END
```

3. Write an ALP in 8086 to perform multiplication of two 8-bit numbers.

```
        ASSUME CS: CODE
        ORG 2000H
CODE    SEGMENT
        MOV SI, 3000H
        MOV AL, [SI]
        INC SI
        MOV BL, [SI]
        MUL BL
        INT 03H
CODE    ENDS
        END
```

4. Write an ALP in 8086 to perform 16-bit by 8-bit division.

```
        ASSUME CS: CODE
        ORG 2000H
CODE    SEGMENT
        MOV SI, 3000H
        MOV AL, [SI]
        INC SI
        MOV AH, [SI]
        INC SI
        MOV BL, [SI]
        DIV BL
        INT 03H
CODE    ENDS
        END
```

5. Write an ALP in 8086 to perform an addition of two 16-bit numbers.

```
        ASSUME CS: CODE
        ORG 2000H
CODE    SEGMENT
START:  MOV SI, 3000H
        MOV AX, [SI]
        INC SI
        INC SI
        MOV BX, [SI]
        ADD AX, BX
        INT 03H
CODE    ENDS
        END
```

6. Write an ALP in 8086 to perform subtraction of two 16-bit numbers.

```
        ASSUME CS: CODE
        ORG 2000H
CODE    SEGMENT
START:  MOV SI, 3000H
```

```
        MOV AX, [SI]
        INC SI
        INC SI
        MOV BX, [SI]
        SUB AX, BX
        INT 03H
CODE    ENDS
        END
```

7. Write an ALP in 8086 to perform multiplication of two 16-bit numbers.

```
        ASSUME CS: CODE
        ORG 2000H
CODE    SEGMENT
START:  MOV SI, 3000H
        MOV AX, [SI]
        INC SI
        INC SI
        MOV BX, [SI]
        MUL BX
        INT 03H
CODE    ENDS
        END
```

8. Write an ALP in 8086 to perform 32-bit by 16-bit division.

```
        ASSUME CS: CODE
CODE    SEGMENT
START:  MOV SI, 3000H
        MOV AX, [SI]
        INC SI
        INC SI
        MOV DX, [SI]
        INC SI
        INC SI
        MOV BX, [SI]
        DIV BX
        INT 03H
CODE    ENDS
        END
```

9. Write an ALP in 8086 to perform BCD addition of two 16-bit numbers.

```
        ASSUME CS: CODE
        ORG 2000H
CODE    SEGMENT
START:  MOV SI, 3000H
        MOV AX, [SI]
        INC SI
        INC SI
        MOV BX, [SI]
        ADD AX, BX
        DAA
        INT 03H
CODE    ENDS
        END
```

10. Write an ALP in 8086 to perform BCD subtraction of two 16-bit numbers.

```
        ASSUME CS: CODE
        ORG 2000H
CODE    SEGMENT
START:  MOV SI, 3000H
        MOV AX, [SI]
        INC SI
        INC SI
        MOV BX, [SI]
        SUB AX, BX
        DAS
        INT 03H
CODE    ENDS
        END
```

Programs involving Logical, Branch and Call instructions

11. Write an ALP in 8086 to perform series addition of N 16-bit numbers.

```
        ASSUME CS: CODE
        ORG 2000H
CODE    SEGMENT
START:  MOV SI, 3000H
        MOV CL, [SI]
        INC SI
        MOV AX, [SI]
        DEC CL
UP:     INC SI
        INC SI
        MOV BX, [SI]
        ADC AX, BX
        DEC CL
        JNZ UP
        INT 03H
CODE    ENDS
        END
```

12. Write an ALP in 8086 to perform subtraction of N 16-bit numbers.

```
        ASSUME CS: CODE
        ORG 2000H
CODE    SEGMENT
START:  MOV SI, 3000H
        MOV CL, [SI]
        INC SI
        MOV AX, [SI]
        DEC CL
UP:     INC SI
        INC SI
        MOV BX, [SI]
        SBB AX, BX
        LOOP UP
        INT 03H
CODE    ENDS
        END
```

13. Write an ALP in 8086 to perform multiplication of given two numbers using
    1. MUL instruction
    2. Repeated addition method

1. MUL instruction

```
        ASSUME CS: CODE
        ORG 2000H
CODE    SEGMENT
        MOV SI, 3000H
        MOV AL, [SI]
        INC SI
        MOV BL, [SI]
        MUL BL
        INT 03H
CODE    ENDS
        END
```

2. Repeated addition method

```
ASSUME CS: CODE
        ORG 2000H
CODE    SEGMENT
        MOV SI, 3000H
        MOV AX, 0000H
        MOV CL, [SI]
        INC SI
UP:     ADC AL, [SI]
        LOOP UP
        INT 03H
CODE    ENDS
        END
```

14. Write an ALP in 8086 to transfer a block of N bytes from one location to another location.

```
        ASSUME CS: CODE
        ORG 4000H
CODE    SEGMENT
        MOV SI, 2000H
        MOV DI, 3000H
        MOV CL, [SI]
UP:     INC SI
        MOV AL, [SI]
        MOV [DI], AL
        INC DI
        LOOP UP
        INT 03H
CODE    ENDS
        END
```

15. Write an ALP in 8086 to exchange a block of N bytes between source location and destination.

```
        ASSUME CS: CODE
        ORG 4000H
CODE    SEGMENT
        MOV SI, 2000H
        MOV DI, 3000H
        MOV CL, [SI]
UP:     INC SI
        MOV AL, [SI]
```

```
        MOV BL, [DI]
        XCHG AL, BL
        MOV [SI], AL
        MOV [DI], BL
        INC DI
        LOOP UP
        INT 03H
CODE    ENDS
        END
```

16. Write an ALP in 8086 to find the maximum number from the given array of N numbers.

```
        ASSUME CS: CODE
        ORG 2000H
CODE    SEGMENT
        MOV SI, 3000H
        MOV CL, [SI]
        INC SI
        MOV AX, [SI]
        DEC CL
UP:     INC SI
        INC SI
        CMP AX, [SI]
        JA DOWN
        MOV AX, BX
DOWN:   LOOP UP
        INT 03H
CODE    ENDS
        END
```

17. Write an ALP in 8086 to find the minimum number from the given array of N numbers.

```
        ASSUME CS: CODE
        ORG 2000H
CODE    SEGMENT
        MOV SI, 3000H
        MOV CL, [SI]
        INC SI
        MOV AX, [SI]
        DEC CL
UP:     INC SI
        INC SI
        CMP AX, [SI]
        JB DOWN
        MOV AX, BX
DOWN:   LOOP UP
        INT 03H
CODE    ENDS
        END
```

18. Write an ALP in 8086 to count no. of even and odd numbers from the given array.

```
        ASSUME CS: CODE
        ORG 2000H
CODE    SEGMENT
        MOV SI, 3000H
        MOV CL, [SI]
        MOV BX, 0000H
        MOV DX, 0000H
```

```
        MOV AX, 0000H
UP:     INC SI
        MOV AL, [SI]
        ROR AL, 01H
        JC ODD
        INC BX
        JMP DOWN
ODD:    INC DX
DOWN:   LOOP UP
        INT 03H
CODE    ENDS
        END
```

19. Write an ALP in 8086 to find no. of positive and negative numbers from the given array.

```
        ASSUME CS: CODE
        ORG 2000H
CODE    SEGMENT
        MOV BX, 0000H
        MOV DX, 0000H
        MOV AX, 0000H
        MOV SI, 3000H
        MOV CL, [SI]
UP:     INC SI
        MOV AL, [SI]
        ROL AL, 01H
        JC NEG
        INC BX
        JMP DOWN
NEG:    INC DX
DOWN:   LOOP UP
        INT 03H
CODE    ENDS
        END
```

20. Write an ALP in 8086 to count no. of 1's and 0's in a given 16-bit number.

```
        ASSUME CS: CODE
        ORG 2000H
CODE    SEGMENT
        XOR AX, AX
        XOR BX, BX
        XOR DX, DX
        MOV SI, 3000H
        MOV CL, 10H
        MOV AX, [SI]
UP:     ROR AX, 01H
        JC ONE
        INC BX
        JMP DOWN
ONE:    INC DX
DOWN:   LOOP UP
        INT 03H
CODE    ENDS
        END
```

21. Write a Recursive program in 8086 to find the sum of first N integers.

```
        ASSUME CS: CODE
        ORG 5000H
CODE    SEGMENT
        MOV SI, 3000H
        MOV CX, [SI]
        CALL ADD
        INT 03H
CODE    ENDS
        END
ADD:    PROC NEAR
        CMP CX, 0000H
        JE EXIT
        ADD AX, CX
        DEC CX
        CALL ADD
EXIT:   RET
        ENDP
```

Evaluation of arithmetic expressions

22. Write an ALP in 8086 to evaluate the following expressions.
    1. $AB - C/D + E$
    2. $\sum_{n=1}^{n} X_n Y_n$

$\underline{1.\ AB - C/D + E}$

```
        ASSUME CS: CODE
        ORG 4000H
CODE    SEGMENT
        MOV SI, 3000H
        MOV AL, [SI]
        INC SI
        MOV BL, [SI]
        MUL BL
        MOV DX, AX
        MOV AH, 00H
        INC SI
        MOV AL, [SI]
        INC SI
        MOV BL, [SI]
        DIV BL
        MOV AH, 00H
        SUB DX, AX
        INC SI
        MOV AL, [SI]
        ADD AX, DX
        INT 03H
CODE    ENDS
        END
```

$\underline{2.\ \sum_{n-1}^{n} X_n Y_n}$

```
ASSUME CS: CODE
        ORG 4000H
CODE    SEGMENT
        MOV SI, 3000H
```

```
        MOV DI, 5000H
        MOV CL, [SI]
        MOV DX, 0000H
UP:     INC SI
        MOV AL, [SI]
        MUL [DI]
        ADD DX, AX
        INC DI
        LOOP UP
        INT 03H
CODE    ENDS
        END
```

Sorting

23. Write an ALP in 8086 to arrange a given array of N bytes in ascending order.

```
        ASSUME CS: CODE
        ORG 4000H
CODE    SEGMENT
        MOV SI, 3000H
        MOV CL, [SI]
        DEC CL
UP1:    MOV CH, [SI]
        DEC CH
        INC SI
UP:     MOV AL, [SI]
        INC SI
        CMP AL, [SI]
        JL OUT
        XCHG AL, [SI]
        XCHG AL, [SI-1]
OUT:    DEC CH
        JNZ UP
        DEC CL
        JNZ UP1
        INT 03H
CODE    ENDS
        END
```

24. Write an ALP in 8086 to arrange a given array of N bytes in descending order.

```
        ASSUME CS: CODE
        ORG 4000H
CODE    SEGMENT
        MOV SI, 3000H
        MOV CL, [SI]
        DEC CL
UP1:    MOV CH, [SI]
        DEC CH
        INC SI
UP:     MOV AL, [SI]
        INC SI
        CMP AL, [SI]
        JG OUT
        XCHG AL, [SI]
```

```
        XCHG AL, [SI-1]
OUT:    DEC CH
        JNZ UP
        DEC CL
        JNZ UP1
        INT 03H
CODE    ENDS
        END
```

Strings

25. Write an ALP in 8086 to insert a byte in to a string.

```
        ASSUME CS: CODE
        ORG 5000H
CODE    SEGMENT
        MOV CX, 0000H
        MOV SI, 3000H
        MOV CL, [SI]
        ADD SI, CX
        MOV DI, SI
        INC DI
        SUB CL, position
        INC CL
        STD
        REP MOVSB
        MOV [DI], Byte
        INT 03H
CODE    ENDS
        END
```

26. Write an ALP in 8086 to check whether the given string is palindrome or not.

```
        ASSUME CS: CODE
        ORG 5000H
CODE    SEGMENT
        MOV CX, 0000H
        MOV SI, 3000H
        MOV CL, [SI]
        MOV DI, SI
        ADD DI, CX
        MOV AL, CL
        MOV BL, 02H
        DIV BL
        MOV CL, AL
        INC SI
UP:     CMPSB
        JNE EXIT
        INC SI
        DEC SI
        LOOP UP
        MOV AX, FFFFH
        INT 03H
EXIT:   MOV AX, 0000H
        INT 03H
```