CHAPTER 3

80x86 INSTRUCTIONS

3.1 INTRODUCTION

The machine language *program code* is the only code that can be executed by the processor. The assembly language code is one of the most primitive forms, in which program can be coded. The assembly language code has to be processed by the assembler to generate machine language code. There are two types of statements in assembly language. First, the instructions, which are translated to the machine code by the assembler, and second, the directives, which direct the assembler during the assembly process for which no machine code is generated.

Assembly language programming is much easier than machine language programming. An instruction in the assembly language is represented by character strings called mnemonics (eg. ADD, SUB, JMP, etc.). The 80x86 processor provides an exhaustive set of instructions to support assembly language programming to perform input, processing, and output operations known as the Instruction Set.

The instructions of 80x86 processor are classified into the following six functional groups.

1. Data Transfer Instructions
2. Arithmetic and Logical Instructions
3. Branch Instructions
4. Processor Control Instructions
5. String Operation Instructions
6. Protection Control Instructions

3.2 ASSEMBLER INSTRUCTION FORMAT

The general format of an assembly language instruction is

```
Label: Mnemonic      Operand, Operand    ; Comment
```

where each part of the instruction is separated by space(s) and every instruction starts on a new line. A label followed by a colon (:) is an identifier that is assigned the address of the first byte of the instruction at the point of occurrence. The use of a label with the instruction is optional. If it is present, the label name can be used in a branch instruction to branch to the labeled instruction. If label is not present, then colon (:) should not to be used. Mnemonic indicates the operation to be performed on the specified operands. The number of operands that can be specified in an instruction depends on the type of instruction. Instructions can operate on an implicit (zero), one, two, or three operands. If there are two or more operands, they should be separated by a comma (,). The comment field is for commenting the program which generally describes the logical operations performed by the instruction. A comment must start with the character semicolon (;). The comment field is optional and is completely ignored by the assembler.

Example:

```
NEXTNUM:
        ADD     AX, AMOUNT          ; Add AMOUNT to the register AX
        ..
        ; write code as required by the program
        ..
        LOOP    NEXTNUM
```

In the above statements, the symbol NEXTNUM is a label, AMOUNT is a variable, and symbols ADD and LOOP are mnemonics of the instructions. All the information followed by the character semicolon (;) are the comments. Assembler generates machine code for only ADD and LOOP instructions including the instructions within the loop.

Following are the conventions used in describing the instructions.

Convention	Meaning
src	source
dest	destination
reg	register
r/m	register or a memory location
r/m16	16 bit register or a word memory location
immed16	immediate 16 bit number
addr	address of a memory location
reg8	8-bit register
reg16	16-bit register
src8	source of type 8-bit
src16	source of type 16-bit
mem16	16-bit memory location
MOD	modulus or a remainder
~	logical NOT operation
&	logical AND operation
|	logical OR operation
^	logical EX-OR (exclusive OR) operation
←	assignment operation
≠	not equal

3.3 DATA TRANSFER INSTRUCTIONS

The 80x86 processor supports a variety of instructions for transfer of data, immediate value, or addresses into registers, memory locations, or ports. The manner in which the operands are designated in the data movement instruction, depends on the addressing modes and can be in any one of the following forms.

- Register to a register
- Immediate operand to a register
- Immediate operand to the memory
- Memory to a register
- Register to a memory
- Register to a segment register (excluding CS)
- Memory to a segment register (excluding CS)
- Segment register to a register
- Register to an I/O port
- I/O port to a register

Data transfer instructions generally involve two operands, the source and destination. The source can be a register or a memory location or an immediate data. The destination can be a register or a memory location. Both the source and destination cannot refer to memory locations in the same instruction. They must be of the same data-type i.e. either of the type byte or type word.

Data transfer instructions do not affect the CPU flags. The various data transfer instructions such as MOV, XCHG, PUSH, POP, IN, OUT , etc., are described in the following section.

MOV : Move

Syntax:

MOV dest , src — Transfer data from register/memory/immediate to register/memory
dest ← src

Description:

The instruction MOV transfers a byte or a word of data from the source to the destination. The source can be a register or a memory location or an immediate number. The destination can be a register or a memory location. Both the source and destination cannot refer to memory locations in the same instruction. They must be of the same data-type, i.e., either of the type byte or type word.

Flags Affected: None

Examples:

```
MOV    AX, CX          ; copy contents of CX to AX
MOV    AH, AL          ; copy contents of AL to AH
MOV    DS, AX          ; copy contents of AX to DS
MOV    AX, 100H        ; copy immediate value 100H to AX
MOV    BX, [530BH]     ; copy 16-bit data from the memory location 530BH
                       ; to BX register
MOV    DL, [BX]        ; copy 8-bit contents of memory location pointed
                       ; to by the address stored in BX, to DL
MOV    BP, SS          ; copy contents of SS to BP
```

XCHG : Exchange

Syntax:

XCHG dest , src — Exchange the contents of the source and destination
dest ←→ src

Description:

The instruction XCHG swaps (exchanges) the contents of a source register or memory with the contents of a destination register or memory. The source can be a register or a memory location. The destination can be a register or a memory location. Both the source and destination cannot refer to memory locations in the same instruction. They must be of the same data-type, i.e., either of the type byte or type word.

Flags Affected: None

Note:The symbol AMOUNT used in the explanation of the following examples in this chapter is considered as an array of type word with size 100 as follows.

```
AMOUNT DW    100 dup(0)
```

Examples:

```
XCHG   AX, CX                ; exchange the contents of AX and CX
XCHG   AL, AMOUNT[BX]        ; exchange(8-bit) AL and memory AMOUNT[BX]
XCHG   AX, AMOUNT[BX]        ; exchange(16-bit) AX and memory AMOUNT[BX]
XCHG   BL, AH                ; exchange BL and AH
XCHG   SEMAPHORE, BX         ; exchange SEMAPHORE with BX
```