8086 PROGRAMMING

ADDRESSING MODES/ASSEMBLERS/DIRECTIVES AND INT10/21 FUNCTIONS

Assembly language programming:

- Needs good knowledge of machine architecture, operating system and programming principles.
- It is case insensitive; compact and efficient which need to be converted into machine code for execution using ***assembler.***
- **Advantages:**
  - Generate compact and small execution module.
  - More control over the hardware.
  - Faster in execution.
- **Disadvantages:**
  - Highly machine dependent
  - Lengthy code
  - Error prone

Features:

- Assembly Language Syntax includes:
  - **Comments:** starts with semi column (;)
  - **Reserved words:**
    - **i.** **Instruction:** such as MOV, ADD etc. (operations to execute)
    - **ii.** **Directives:** such as END, SEGMENT (information's to the assembler)
    - **iii.** **Operators:** FAR, SIZE etc. (used in expressions)
    - **iv.** **Predefined symbols:** @DATA, @MODEL etc. (returns information to the program during assembly)
  - **Identifiers:** Two types of identifiers are ***Name*** and ***Label***.
  - **Statements:** ALP consists of a set of statements with two types:
    - i. Instructions, e. g. MOV, ADD
    - ii. Directives, e. g. define a data item
  - **Directives:** Number of statements that enables us to control the way in which the source program assembles and lists.
  - **Operators:** Mathematical, logical and other symbols to facilitate operations.
- which provide the basic rule and frame work for the language.

Assembly Language Instruction Format:

Assembly language: Op-codes, mnemonics and operands

- Represented by certain words representing the operation of instruction. Thus programming gets easier.
- Generally written in a standard form that has four fields.
  - Label field
  - Op-code field (instruction or mnemonic)
  - Operand field
  - Comment field

Example:

Label	Mnemonic	Operand	Comment
Start:	MVI	A,10H	; Move 10H in register A
	MVI ADD HLT	B,04H B	; Move 04H in register B ; Add the content of register B with that of A ; Stop program

Mnemonic:

- short alphabetic code used in assembly language for microprocessor operation.
- words (usually two-to-four letter) used to represent each instruction.

Assembler:

- software (program module) which converts assembly language code (source module) into a machine language code.
- Assembly language program are called “source codes”.
- Machine language programs are known as object codes.

Assembler types:

- **One Pass Assembler**:
  - Goes (scans) through the program once
  - Can resolve backward reference
  - Cannot resolve forward reference
- **Two Pass Assembler**:
  - Goes (scans) through the assembly language program twice.
  - First pass generates the table of the symbol, which consists of labels with addresses assigned to them.
  - Second pass uses the symbol table generated in the first pass to complete the object code for each instruction thus producing machine code.
  - It can resolve forward and backward reference both.

Backward reference	**Forward reference**
L1: …..	JMP L1
…..	…..
…..	…..
…..	…..
JMP L1	L1: …..

Linker:

- Program that links object file created by assembler into a single executable file.
- A translator converts source codes to object codes and then into executable formats.
- Converting source code into object code is called compilation and assembler does it.
- Converting object codes into executable formats is called linking and ***linker*** does it.

*.asm ~~assembler~~ *.obj ~~linker~~ *.exe

Macro assembler:

- an assembly language that allows macros to be defined and used.
- The Microsoft Macro Assembler (MASM) is an x86 assembler that uses the Intel syntax for MS-DOS and Microsoft Windows.
- translates a program written in macro language into the machine language.
- A macro language is the one in which all the instruction sequence can be defined in macro block.
- A macro is an instruction sequence that appears repeatedly in the program assigned with specific name.
- The MASM replaces a macro name with appropriate instruction sequence whenever it encounters a macro name.

- E.g.

```
Initiz macro
Mov ax, @dataseg
Mov ds, ax
Mov es, ax
Endm

 OR,
Initiz {
Mov ax, @dataseg
Mov ds, ax
Mov es, ax
}
```

- ***There exists little difference between macro program and subroutine program.***

- **Subroutine program:**
  - execution jumps out of main program and executes subroutine and control returns to the main program after RET instruction.

- **Macro:**
  - Does not cause program execution to branch out of main program.
  - Each time a macro occurs, it is replaced with appropriate sequence in the main program.

- **Advantages of using Macro:**
  - To simplify and reduce the repetitive coding.
  - To reduce errors caused by repetitive coding.
  - To make assembly language program more readable.

ASSEMBLER DIRECTIVES

- Instruction that will give the information to assembler for performing assembling are called assembler directives.
- Not executed by MP, so they are called dummy or pseudo instructions.
- It gives direction to the assembler.

8086 commonly used directives

- **PAGE and TITLE directive: Format:** PAGE [Length] [Width]

  e.g., **PAGE** [60] [90] means length of the page is of 60 lines and each line can contain 90 characters.

  Default: Page [50] [80]

  Format: TITLE text [comment]

- **Segment directive:**

  Format: Segment-name Segment [align][combine][‘class’]

  …….

  Segment-name ENDS

- **Proc directive: Format:** PROC – name PROC [FAR/NEAR]

  …….

  …….

  PROC - name ENDP

- **END directive: Format:** END PROC-Name
- **ASSUME directive: Format:** Assume SS: Stack name, DS: Data Seg-name CS: Code Seg-name
- **EQU directive:** E.g. **FACTOR** EQU **12**

  MOV BX, FACTOR ; equivalent to MOV BX, 12
- **DUP directive Data Definition directive:**
  - Used to initialize several locations to zero. E. G. SUM DW 4 DUP(0) reserves four words starting at the offset sum in DS and initializes them to zero.
  - Also used to reserve several locations that need not be initialized. In this case (?) is used with DUP directives.

    E. g. **PRICE** DB 100 DUP(?) Reserves 100 bytes of uninitialized data space to an offset PRICE.
- **ORG directive**
- **Macro and Endm directive**
- **OFFSET directive**

Simplified Segment Directives

Memory Model	Description
Tiny	Code and data together may not be greater than 64K
Small	Neither code nor data may be greater than 64K
Medium	Only the code may be greater than 64K
Compact	Only the data may be greater than 64K
Large	Both code and data may be greater than 64K
Huge	All available memory may be used for code and data

- **. stack**
  - The .stack directive sets the size of the program stack, which may be any size up to 64K. This segment is addressed by SS and SP registers.

- **. code**
  - The .code directive identifies the part of the program that contains instructions. This segment is addressed by CS and IP registers.

- **. data**
  - All variables are defined in this segment area.

Addressing modes of 8086:

- There are 8 different addressing modes in 8086 programming:

1. ***Immediate addressing mode***

   The addressing mode in which the data operand is a part of the instruction itself is known as immediate addressing mode.
   Example: MOV CX, 4929H, ADD AX, 2387H, MOV AL, FFH

2. ***Register addressing mode***

   The register is the source of an operand for an instruction.
   Example: MOV CX, AX; copies the contents of the 16-bit AX register into the 16-bit CX register),
   ADD BX, AX

3. *Direct addressing mode*

The addressing mode in which the effective address of the memory location is written directly in the instruction.

Example: MOV AX, [1592H],
MOV AL, [0300H],
ADD AX, [7765H]

4. *Register indirect addressing mode*

This addressing mode allows data to be addressed at any memory location through an offset address held in any of the following registers: BP, BX, DI & SI.

Example: MOV AX, [BX]; Suppose the register BX contains 4895H, then the contents of 4895H are moved to AX

ADD CX, [SI]

5. ***Based (displacement) addressing mode***

In this addressing mode, the offset address of the operand is given by the sum of contents of the BX/BP registers and 8-bit/16-bit displacement.

Example: MOV DX, [BX+04],
ADD CL, [BX+08]

6. ***Indexed addressing mode***

In this addressing mode, the operands offset address is found by adding the contents of SI or DI register and 8-bit/16-bit displacements.

Example: MOV BX, [SI+16],
ADD AL, [DI+16]

Operators in 8086:

A. **Arithmetic Operators:** + (Addition), - (Subtraction), + (Positive), - (Negation), * (Multiplication), / (Division) and % (Remainder).

B. **Index operators:** Used in indirect addressing of memory, reference to base or index register, contents, offsets and variables. Notation used is square bracket [ ]. *E.g. MOV AX, [BX]*

C. **Logical operators:** used for bit manipulation, OR, AND, NOT, XOR etc.
D. **Shift operators:** SHR, SHL
E. **Macro operator**
F. **Record operators:** Generally, two types in 8086:
   - **MASK:** Return a mask of 1-bit representing the specified field and defines the bit positions that the field occupies.
   - **WIDTH:** Returns a width as the number of bits in a RECORD.
G. **Relational operators:** The relational operators are: EQU, GE, GT, LE, LT, NE
H. **Segment operators:** OFFSET, SEG, segment override
I. **Type (Attribute) operator:** HIGH,LOW, PTR,TYPE,LENGTH,SHORT ETC.

DOS functions and Interrupts

Keyboard and video processing

INT 21h functions

- Provided in the photocopy
- DOS services for keyboard operations
- 01h, 02h, 09h, 0Ah, 4Ch

INT 10h functions

- Provided in the photocopy
- Video display services (BIOS services)
- Controls screen format, color, text style, making window, scrolling etc.
- 00h, 01h, 02h, 06h, 07h, 08h, 09h, 0Ah

INT 21h

Function number	Description
01h e.g. mov ah,01h int 21h	Keyboard input with echo: This operation accepts a character from the keyboard buffer. If none is present, waits for keyboard entry. It returns the character in AL.
02h e.g. mov ah,02h int 21h	Display character: Send the character in DL to the standard output device console.
09h e.g. mov ah, 09h Int 21h	String output: Send a string of characters to the standard output. DX contains the offset address of string. The string must be terminated with a „$“ sign.
0Ah	String input
4Ch e.g. mov ax,4C00h int 21h	Terminate the current program.

INT 10h

Function number	Description
00h	Set video mode
01h	Set cursor size
02h	Set cursor position
06h	Scroll window up
07h	Scroll window down
08h	Read character and attribute of cursor
09h	Display character and attribute at cursor
0Ah	Display character at cursor