LECTURE # 05

PREPARED BY: TAHMINA KHAN

ASSEMBLY LANGUAGE PROGRAMMING

- The components of a simple Assembly language program to be assembled by the assembler.
- A series of statements or lines which are either assembly language instructions such as MOV and ADD, or statements called *directives*.
- *Directives* (also called pseudo-instructions) give directions to the assembler about how it should translate the Assembly language instructions into machine code.
- An assembly language instructions consists of four fields:

[label:] mnemonic [operands] [;comments]

Note: Labels for directives do not need to end with a colon. A label must end with a colon when it refers to an opcode generating instruction; the colon indicates to the assembler that this refers to code within this code segment.

MODEL DEFINITION

MODEL directive selects the size of memory model.

.MODEL SMALL
- Most widely used MEMORY MODEL.
- The code must fit in 64k.
- The data must fit in 64k.

.MODEL MEDIUM
- The code can exceed 64k.
- The data must fit in 64k.

.MODEL COMPACT
- The code must fit in 64k.
- The data can exceed 64k.

.MODEL LARGE
- Both code and data can exceed 64k.
- No single set of data can exceed 64k.

.MODEL HUGE
- Both code and data can exceed 64k.
- A single set of data can exceed 64k.

.MODEL TINY
- Used with COM files.
- Both code and data must fit in a single 64k segment.

SEGMENT DEFINITION

- The CPU has several segment registers:
  - Code segment
  - Data segment
  - Stack segment
  - Extra segment
- Every instruction and directive must correspond to a segment.
- Normally a program consists of three segments: the stack, the data and the code segment.
- Segment definition formats:
  - Simplified segment definition
  - Full segment definition

SEGMENT DEFINITION

- The simplified segment definition uses the following directives to define the segments:

  .STACK

  .DATA

  .CODE

- These directives mark the beginning of the segments they represent.

SEGMENT DEFINITION

- The full segment definition uses the following directives to define the segments:

```
Label SEGMENT
        ; statements belonging to the segment
Label ENDS
```

SEGMENT DEFINITION

```
;SIMPLIFIED SEGMENT DEFINITION

.MODEL SMALL

.STACK 64



.DATA
N1      DW      1432H
N2      DW      4365H
SUM     DW      0H


.CODE
BEGIN   PROC FAR

        MOV AX,@DATA
        MOV DS,AX
        MOV AX,N1
        ADD AX,N2
        MOV SUM,AX
        MOV AH,4CH
        INT 21H
BEGIN   ENDP
        END BEGIN
```

```
;FULL SEGMENT DEFINITION



STSEG   SEGMENT
DB 64 DUP(?)
STSEG   ENDS

DTSEG   SEGMENT
N1      DW      1432H
N2      DW      4365H
SUM     DW      0H
DTSEG   ENDS

CDSEG SEGMENT
BEGIN   PROC FAR
        ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
        MOV AX,DTSEG
        MOV DS,AX
        MOVAX,N1
        ADD AX,N2
        MOV SUM,AX
        MOV AH,4CH
        INT 21H
BEGIN   ENDP
CDSEG   ENDS
        END BEGIN
```

DATA TYPES

The assembler supports all the various data types of the x86 microprocessor by providing data directives that define the data types and set aside memory for them. The following are some of the data directives used by the x86 microprocessor.

- DB (define byte)
- DW (define word)
- DD (define double word)
- DQ (define quad word)
- DT (define ten bytes)
- ORG (origin)
- DUP (duplicate)
- EQU (equate)

DATA TYPES

- **DB (define byte)**

The DB directive is one of the most widely used data directive in the assembler. It allows allocation of memory in byte sized chunks. The following are some examples of DB:

```
D1      DB 24               ; Decimal
D2      DB 10100100B        ; Binary
D3      DB 5FH              ; Hex
```

- **DW (define word)**

The DW directive is used to allocate 2 bytes (one word) at a time. The following are some examples of DW:

```
D1      DW 954              ; Decimal
D2      DW 100101010100B    ; Binary
D3      DW 253FH            ; Hex
```

DATA TYPES

- **DD (define double word)**

The DD directive is used to allocate memory locations that are 4 bytes (two words) in size. The data can be placed in memory location according to the rule of low byte to lower address and high byte to higher address. The following are some examples of DD:

```
D1      DD 5C2A57F2H                ; Hex
D2      DD 1023                     ; Decimal
D3      DD 10101010101010101010B    ; Binary
```

- **DQ (define quad word)**

The DQ directive is used to allocate memory 8 bytes (four words) in size. This can be used to represent any variable upto 64 bits wide. The following are some examples of DQ:

```
D1      DQ 452A57H          ; Hex
D2      DQ 'HI'             ; ASCII
D3      DQ ?                ; Nothing
```

DATA TYPES

- **DT (define ten bytes)**

DT is used for memory allocation of packed BCD numbers. The application of DT will be in the multibyte addition of BCD numbers. This directive allocates 10 bytes , but a maximum of 18 digits can be entered. The following are some examples of DT:

```
D1      DT 864973569829        ; BCD
D2      DT ?                   ; Nothing
```

- **ORG (origin)**

ORG is used to indicate the beginning of the offset address. The number that comes after ORG can be either in hex or in decimal. If the number is not followed by H, it is decimal and the assembler will convert it to hex. The following are some examples of ORG:

```
ORG 0010H
ORG 38H
ORG CAH
```

DATA TYPES

- **DUP (duplicate)**

DUP is used to duplicate a given number of characters. This can avoid a lot of typing. For example, the method of filling six memory locations with FFH:

```
DATA1  DB 6 DUP (0FFH)          ; Fill 6 bytes with FF
```

PROCEDURE

- A procedure is a group of instructions designed to accomplish a specific function.
- A code segment may consist of only one procedure, but usually is organized into several small procedures in order to make the program more structured.
- Every procedure must have a name defined by the PROC directive, followed by the assembly language instructions and closed by the ENDP directive.
- The PROC and ENDP statements must have the same label.

PROCEDURE

- The PROC directive may have the option FAR or NEAR.
- If control is transferred to a memory location within the current code segment, it is NEAR or intrasegment (within segment).
- If control is transferred outside the current code segment, it is FAR or intersegment (between segments).

ASSEMBLE, LINK & RUN A PROGRAM

- The three steps to create an executable Assembly language program are outlined as follows:

Table 1: Program Input and Output

Step	Input	Program	Output
1. Edit the program	Keyboard	Editor	myfile.asm
2. Assemble the program	myfile.asm	MASM or TASM	myfile.obj
3. Link the program	myfile.obj	LINK or TLINK	myfile.exe