The use of assembler directives in microprocessor

Abstract

This term paper includes the use of assembler directives in microprocessor and the various assembler directives used in Intel microprocessor 8086, its format with various examples.

I) Definition

An assembler directive is a message to the assembler that tells the assembler something it needs to know in order to carry out the assembly process; for example, an assemble directive tell the assembler where a program is to be located in memory.

There are some instructions in the assembly language program which are not a part of processor instruction set. These instructions are instructions to the assembler, linker and loader. These are referred to as pseudo-operations or as assembler directives.

II) Use Of Assembler Directives

The assembler directives control organization of the program and provide necessary information to the assembler to understand assembly language programs to generate machine codes. They indicate how an operand or a section of program is to be processed by the assembler. An assembler supports directives to define data, to organize segments to control procedures, to define macros etc. An assembly language program consists of two types of statements: instructions and directives. The instructions are translated to machine codes by the assembler whereas the directives are not translated to machine codes.

III) Assembler Directives For Intel8086 Microprocessor

1) ASSUME

The ASSUME directives is used to inform the assembler the names of the logical segments, which are to be assigned to the different segments used in an assembly language program. In an assembly language program, each segment is given a name by the programmer. For example, the code segment may be given the name CODE or CODE_SEG or MY_CODE, etc. The data segment may be given the name DATA, MY_DATA, DATA_SEG, etc.

Examples:

i) ASSUME CS : CODE

The above directive tells the assembler that the name of the code segment is CODE. This name is a user defined segment name. The segment named CODE contains the machine codes of the instructions. The code segment register (CS register) is to be loaded with the starting address of the code segment, given by the operating system for the label CODE in the assembly language program.

ii) ASSUME DS: DATA

The above directive informs the assembler that the name of the data segment is DATA. This is a user defined segment name. It contains data of the program which is being executed. The DS register (data segment register) is to be loaded with the starting address of the data segment, given by the operating system for the label DATA in the program.

iii) ASSUME SS: STACK

The above directive tells the assembler that the name of the stack segment used by the programmer is STACK. This is a user defined segment name. It stores addresses and data of the subroutines, saves the contents a specified register or memory locations after PUSH instruction, etc. The stack segment register SS holds the starting address of the stack segment allotted by the operating system.

iv) ASSUME ES: EXTRA

The above directive tells the assembler that the name of the extra segment is EXTRA which is a user defined segment name. In Intel 8086 microprocessor, string instructions may use DI register to point the destination memory address for the data. The EXTRA segment is used to hold the addresses pointed by DI.

2) DB (Define Byte)

The directive DB a byte type variable. In a given directive statement, there may be single initial value or multiple initial values of the defined variable. If there is one initial value, one byte of memory space is reserved for each value. The general format is:

Name of Variable DB Initial value or Values.

Examples:

i) VELOCITY DB 0

This directive informs assembler to reserve one byte of memory space for the variable named VELOCITY and initialize it with value zero.

ii) WEIGHT DB 85

This directive informs assembler to reserve one byte of memory space for the variable named WEIGHT and initialize with value 85.

iii) FORCE DB ?

This directive directs assembler to reserve one byte of memory space for the variable FORCE. Furthermore, the question mark ? In the data definition informs assembler that the value of the variable is not known and hence, it is not to be initialized.

iv) ARRAY DB 32, 42, 59, 67, 83

This directive informs assembler to reserve five bytes of consecutive memory space for the variable named ARRAY. The memory locations are to be initialized with the values 32, 42, 59, 67 and 83.

v) MESSAGE DB ‘ THANK YOU’

This directive informs the assembler to reserve the number of bytes of memory space equal to the number of characters in the string named MESSAGE, and initialize the memory locations with ASCII codes of the these characters.

3) DW (Define Word)

The directive DW defines a word -type variable. The defined variable may have one or more initial values in the directive statement. If there is one value, two-bytes of memory space are reserved. If there are multiple values, two bytes of memory space are reserved for each value. The general formula is:

Name of variable DW Initial Value or Values.

Examples:

i) SUM DW 3596.

This directive informs the assembler to reserve two bytes (one word) of consecutive memory locations for the variable named SUM and initialize it with the value 3596.

ii) NUMBER DW ‘ 25′

The above directive statement informs assembler to reserve two bytes of consecutive memory locations for the variable named NUMBER. The first byte of the memory is to be initialized with the ASCII code of two (32) and the second byte is to be initialized with the ASCII code of five (35). Hence, the two bytes of memory space contain 3235H.

iii) DATA DW 5384, 6932, 5 DUP (3456), 7384

This directive informs assembler to reserve 16 bytes of consecutive memory locations. The number 3456 is repeated five times. Memory locations are initialized with 5384, 6932, 3456, 3456, 3456, 3456, 3456 and 7384.

4) DD (Define Double Word)

This directive DD defines a double word-type variable. The defined variable may have one or more values in the statement. If there is only one value, four bytes of consecutive memory locations are reserved. If there are multiple values, four bytes of memory locations are reserved for each value. The general format is:

Name of Variable DD Initial value or values

Example:

NUMBER DD 23958634

The above directive informs assembler to reserve four bytes of memory locations for the variable named NUMBER and initialize with the number 23958634.

5) DQ (Define Quadword)

The directive DQ defines a quadword- type variable. The defined variable may have one or more values in the statement. If there is only one value, 8 bytes of consecutive memory locations are reserved. If there are multiple values, 8 bytes of memory space are reserved for each value. The general format is:

Name of Variable DQ Initial value or values

Example:

NUMBER DQ 1568934893846735

The above directive informs assembler to reserve 8 bytes of consecutive memory locations for the variable named NUMBER and initialize with the above mentioned number.

6) DT (Define Tenbytes)

The directive DT defines a variable of ten bytes. In the directive statement there may be one or more values. If there is only one value, 10 bytes of consecutive memory locations are reserved. If there are multiple values, ten consecutive memory locations are reserved for each value. The general format is:

Name of Variable DT Initial value or values

Example:

NUMBER DT 34968435876934839251

The above directive informs assembler to reserve 10 bytes of consecutive memory locations for the variable named NUMBER and initialize with the above specified values.

7) END (End of Program)

The directive END informs assembler the end of a program module. This is used after the last statement of the program module. This assembler ignores statement(s) after an END directive. Therefore, the programmer should use END directive at the very end of his program module. A carriage return is used after the END directive. Its general format is:

END label

8) ENDP (End Procedure)

The directive ENDP informs assembler the end of a procedure. In assembly language programming, subroutines are called procedures. A procedure may be an independent program module to give certain result or the required value to the calling program. A procedure is given a name i. e. a label. The label is used as prefix with directive ENDP. This directive is used together with PROC directive to enclose the procedure. To specify the type of the procedure the term FAR or NEAR is used after the PROC directive. The type FAR indicates that the procedure to be called is in some other segment of memory. Type NEAR indicates that the procedure is in the same segment of memory. If type is not specified, the assembler assumes it NEAR. The general format for ENDP directive is:

Procedure Name ENDP

Example:

SPEED_CONTROL PROC FAR ; Start of Procedure

: ; Procedure instructions

SPEED_CONTROL ENDP ; End of Procedure

9) ENDM (End Macro)

The directive ENDM is used to inform assembler that this is the end of a macro. The directive ENDM is used with the directive MACRO to enclose macro instructions.

Example:

COMPLIMENT MACRO ; Start of macro

: ; Macro instructions

ENDM ; End of Macro

COMPLIMENT is the name of a macro. The name is given before the directive MCRO which tells the assembler the beginning of a macro.

10) ENDS (End of Segment)

The ENDS directive informs assembler that this is the end of the segment. The name of the segment is given using ASSUME directive which has already been explained. The name of the segment is used as the prefix of the ENDS directive. Its general format is:

Segment Name ENDS

Example:

CODE_SEG SEGMENT ; Start of code segment

– ; instructions

CODE_SEG ENDS ; End of segment

11) EQU (Equate)

The directive EQU is used to give a name to certain value or symbol. If any value or symbol is used many times in an assembly language program, a name is given to the value or symbol to make programming easier and simpler. Each time the assembler finds the given name in the program, it replaces the name with the value or symbol which has already been equated with that name. The general format for the EQU directive is:

Value Name EQU Value

or or

Symbol Name Symbol

Example:

ROOM_TEMP EQU 02H

The above directive tells assembler to replace ROOM_TEMP by 02H. If EQU directive is written in the beginning of the program and later on MOV AL, ROOM_TEMP is written in the program, the assembler will treat this instruction as if it were MOV AL, 02H while giving its machine codes.

12) EXTRN (External)

This directive informs the assembler that the names, procedures and labels following this directive have already been defined in some other program modules. The names, procedures and labels declared as external in one program module must be declared public using PUBLIC directive in the

program module in which they have been defined. When the programmer informs assembler that the declared item is an external one, the assembler puts this information in the object code file so that the linker can connect the concerned two program modules together. The general format for EXTRN directive is:

i) EXTRN Variable Name : Type of variable

ii) EXTRN Procedure Name : (NEAR/FAR)

For external named variable, procedure or constant; its type is to be specified.

Examples :

i) EXTRN MULTIPLIER : WORD

In this directive the variable named MULTIPLIER is an external variable and it is word type variable.

ii) EXTRN CORRECTION_FACTOR: ABS

In this directive CORRECTION_FACTOR is an external constant. It has been defined with EQU directive in another program module. Constants are identified by type: ABS.

13) LABEL (Label)

In an assembly language program labels are used to give names to memory addresses. When assembler begins assembly process, it initializes a location counter to keep the track of memory locations i. e. memory addresses. The

content of the location counter holds the address of the memory location assigned to an instruction during assembly process. The LABEL directive is used to give a name to the current value in the location counter i. e. the current memory address which is in the location counter. The type of label is to be specified. The general format of the LABEL directive is:

LABEL Label Name Label Type

Example:

AHEAD LABEL NEAR

Instruction

Instruction AHEAD

AHEAD Instruction

14) LENGTH (Length)

It is an operator to determine the number of elements in a data item such as an array or a string.

Example:

DATA SEGMENT

ARRAY DW 10 DUP(?)

NUMBERS DB 10 DUP(?)

DATA ENDS

The LENGTH operator can be used as follows:

i) MOV CX, LENGTH ARRAY

ii) MOV CX, LENGTH NUMBERS

In both examples i) and ii), the execution of the instruction will move number 10 in the register CX as the number of elements are 10 in both the variables ARRAY as well as NUMBERS.

15) PROC (Procedure)

The directive PROC indicate the start of a procedure. The type of the procedure FAR or NEAR is to be specified after the directive. The type NEAR is used to call a procedure which is within the program module. The type FAR is used to call a procedure from some other program module. The PROC directive is used with ENDP directive to enclose a procedure. The general format of the PROC directive is:

Name of procedure PROC type of procedure

Example :

i) TEMP_MEAST PROC FAR

The above procedure is for temperature measurement and it lies in some other program module.

There are other directives also which includes TYPE (Type), STRUCT OR STRUC (Structure Declaration), SIZE (Size), SHORT (Short), SEG (Segment),

RECORD (Record), PUBLIC (Public), MACRO, NAME (Name), OFFSET (Offset) etc.