


STRAWBERRY



 /strawberrydevelopers

 /strawberry_app

For more visit:

Strawberrydevelopers.weebly.com

ASSEMBLER DIRECTIVES

Richa Upadhyay Prabhu

NMIMS's MPSTME

richa.upadhyay@nmims.edu

January 12, 2016

The words defined in this section are directions to the assembler, not instructions for the 8086.

ASSUME

- It is used to tell the assembler that the name of the logical segment should be used for a specified segment.
- Works directly with only 4 physical segments: a Code segment, a data segment, a stack segment, and an extra segment.

Example:

- **ASSUME CS:CODE** ; This tells the assembler that the logical segment named CODE contains the instruction statements for the program and should be treated as a code segment.
- **ASSUME DS:DATA** ; This tells the assembler that for any instruction which refers to a data in the data segment, data will found in the logical segment DATA.

DB: DEFINE BYTE

- DB directive is used to declare a byte-type variable or to store a byte in memory location

EXAMPLE:

- PRICE DB 49h, 98h, 29h ; Declare an array of 3 bytes
- NAME DB ABCDEF ; Declare an array of 6 bytes and initialize with ASCII code for letters
- TEMP DB 100 DUP(?) ; Set 100 bytes of storage in memory and give it the name as TEMP, but leave the 100 bytes uninitialized. Program instructions will load values into these locations.

DW: DEFINE WORD

- It is used to define a variable of type word i.e. assembler reserves the no. of memory words 16-bits.

EXAMPLE:

- MULTIPLIER DW 437Ah
- EXP1 DW 1234h, 3456h, 5678h
- STOR1 DW 100 DUP(0) ; Reserve an array of 100 words of memory and initialize all words with 0000. Array is named as STOR1.

DD: DEFINE DOUBLEWORD

Example, TEMP DD 25629261h

DQ : DEFINE QUADWORD ; reserve 4 words of storage in memory

END : End of program

ENDP : End of Procedure (Sub programs)

- procedures are usually given a name i.e. LABEL
- PROCEDURE ADD

.
. .
.

ADD ENDP

ASSEMBLER DIRECTIVE

ENDS : End of Segment

- Marks end of logical segment

```
DATA SEGMENT
```

```
.  
. .  
. .
```

```
DATA ENDS
```

```
ASSUME CS: CODE,DS: DATA
```

```
CODE SEGMENT
```

```
.  
. .  
. .
```

```
CODE ENDS
```

```
END
```

EQU : Equate

- This EQU directive is used to give a name to some value or to a symbol
- Each time the assembler finds the name in the program, it will replace the name with the value or symbol you given to that name.

Example:

FACTOR EQU 03H ;

ADD AL, FACTOR ; When it codes this instruction the assembler will code it as **ADD AL, 03H**

EXTRN : External and PUBLIC

- The EXTRN directive informs the assembler that the names, procedures and labels declared after this directive have already been defined in some other assembly language module.
- In modules where the names, procedures and labels actually appear, they must be declared public using PUBLIC directive.

GROUP : Group the related segment

- Example ;
PROGRAM GROUP CODE,DATA,STACK
- The above statement directs the loader/linker to prepare an EXE file such that the CODE, DATA and STACK segment must lie within a 64 kb memory segment and is named as PROGRAM.
- Now ASSUME statement, one can use the label PROGRAM
ASSUME CS: PROGRAM,DS: PROGRAM,SS: PROGRAM

ASSEMBLER DIRECTIVE

LABEL :Used to assign a name to the current content of the location counter

LENGTH : Determine number of elements in some named data item, such as string or array

ORG - ORIGINATE : Allows to set location counter to desired value (which by default is 0000h at start of program) at any point in program

SEGMENT : Indicate start of a logical segment

Let's have a look at the instruction set :

DATA TRANSFER INSTRUCTIONS

General purpose byte or word transfer instructions:

MOV

PUSH

POP

XCHG

XLAT

Input and output port instructions

IN

OUT

Special Address Transfer

LEA

LDS

LES

Flag transfer instructions

LAHF

SAHF

PUSHF

ARITHMETIC INSTRUCTIONS

Addition Instructions

ADD

ADC

INC

AAA

DAA

Subtraction Instruction

SUB

SBB

DEC

NEG

CMP

AAS

DAS

ARITHMETIC INSTRUCTIONS

Multiplication Instruction

MUL

IMUL

AAM

Division instruction

DIV

IDIV

AAD

CBW

CWD

BIT MANIPULATION INSTRUCTIONS

Logical : NOT, AND, OR, XOR, TEST

Shift : SHL/SAL, SHR

Rotate : ROL, ROR, RCL, RCR

PROGRAM EXECUTION TRANSFER INSTRUCTIONS

Unconditional transfer : CALL, RET, JMP

Conditional transfer : JA/JNBE, JC ,JNC, JS, etc

Iteration control : LOOP, LOOPZ, LOOPNZ

Interrupt Instructions : INT, INTO, IRET

PROCESSOR CONTROL INSTRUCTIONS

Flag set/clear instructions :

STC, CLC,CMC,STD,CLD,STI,CLI

External hardware sync. instruction

HLT, WAIT, ESC, LOCK

No operation instruction

NOP

Implementing Standard Program Structure

Setting up data structures :

- Will data be in memory or in registers?
- Datatype: byte, word, double word ?
- no. of data items.
- data is signed or unsigned ?

Implementing Standard Program Structure

Consider a PROBLEM STATEMENT :

Write an assembly language program for addition of two 8-bit numbers.

Implementing Standard Program Structure

Consider a PROBLEM STATEMENT :

Write an assembly language program for addition of two 8-bit numbers.

Data Structure :

VAR1 DB 85H

VAR2 DB 32H

RES DB ?

Implementing Standard Program Structure

Define data structure in DATA SEGMENT, use SEGMENT and END directives

DATA SEGMENT

VAR1 DB 85H

VAR2 DB 32H

RES DB ?

DATA ENDS

Implementing Standard Program Structure

use ASSUME directive

DATA SEGMENT

VAR1 DB 85H

VAR2 DB 32H

RES DB ?

DATA ENDS

ASSUME CS:CODE,DS:DATA ; logical segment named CODE contains instructions and should be treated as code segment

Implementing Standard Program Structure

Begin writing code in the CODE SEGMENT and also use directive START and END START while writing.

DATA SEGMENT

VAR1 DB 85H

VAR2 DB 32H

RES DB ?

DATA ENDS

ASSUME CS:CODE,

DS:DATA

CODE SEGMENT

START :

write code here

CODE ENDS

END START

Implementing Standard Program Structure

Initialize Data segment register :

```
DATA SEGMENT
```

```
VAR1 DB 85H
```

```
VAR2 DB 32H
```

```
RES DB ?
```

```
DATA ENDS
```

```
ASSUME CS:CODE,
```

```
CODE SEGMENT
```

```
START :
```

```
DS:DATA
```

```
MOV AX,DATA
```

```
MOV DS,AX
```

```
write code here
```

```
CODE ENDS
```

```
END START
```

Implementing Standard Program Structure

Initialize Data segment register :

```
MOV AX,DATA
```

```
MOV DS,AX
```

These instructions load DS register with the upper 16 bits of the starting address for the data segment.

Implementing Standard Program Structure

Start writing the logic :

Implementing Standard Program Structure

Start writing the logic :

```
DATA SEGMENT
```

```
VAR1 DB 85H
```

```
VAR2 DB 32H
```

```
RES DB ?
```

```
DATA ENDS
```

```
ASSUME CS:CODE,
```

```
CODE SEGMENT
```

```
START :
```

```
DS:DATA
```

```
MOV AX,DATA
```

```
MOV DS,AX
```

```
MOV AL,VAR1
```

```
MOV BL,VAR2
```

```
ADD AL,BL
```

```
CODE ENDS
```

```
END START
```

Implementing Standard Program Structure

Knowing DOS (Disk Operating System) Function Calls :

DOS and BIOS (Basic I/O System) are used by assembly language to control the personal computer. The function calls control everything from reading and writing disk data to managing the keyboard and displays.

In order to use function call, always place function number in register AH and load other information into registers

Following is INT 21H : software interrupt to execute a DOS function

Implementing Standard Program Structure

Knowing DOS (Disk Operating System) Function Calls :

TERMINATE PROGRAM AND RETURN TO DOS (DOS FUNCTION 4CH)

MOV AH,4CH
INT 21H

A code of 00H in the AL register indicates normal program termination. Thus the function is usually invoked as:

MOV AX , 4C00H
INT 21H

Implementing Standard Program Structure

DATA SEGMENT

```
VAR1 DB 85H  
VAR2 DB 32H  
RES DB ?
```

DATA ENDS

ASSUME CS:CODE,

DS:DATA

CODE SEGMENT

START :

```
MOV AX,DATA  
MOV DS,AX  
MOV AL,VAR1  
MOV BL, VAR2  
ADD AL,BL  
MOV AH,4CH  
INT 21H
```


CODE ENDS

END START

STRAWBERRY



 /strawberrydevelopers

 /strawberry_app

For more visit:

Strawberrydevelopers.weebly.com