LAB 03

DATA TYPES & ASSEMBLY INSTRUCTIONS

_____   _____   ___
STUDENT NAME   ROLL NO   SEC

_____
SIGNATURE & DATE

MARKS AWARDED: _____

_____
NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES (NUCES), KARACHI

Lab Session 03: DATA TYPE & ASSEMBLY INSTRUCTIONS

Objectives:

- Defining Data
- Data Definition Statement
- Data Initializations
- Multiple Initializations
- String Initialization
- Assembly language Instructions: MOV , ADD , SUB
- Sample Program
- Exercise

Data Types:

MASM defines **intrinsic data types**, each of which describes a set of values that can be assigned to variables and expressions of the given type.

BYTE	8-bit unsigned integer
SBYTE	8-bit signed integer. S stands for signed
WORD	16-bit unsigned integer
SWORD	16-bit signed integer
DWORD	32-bit unsigned. D stands for double
QWORD	64-bit integer. Q stands for quad
TBYTE	80-bit integer. T stands for ten

Data definition statement:

A data definition statement sets aside storage in memory for a variable, with an optional name. Data definition statements create variables based on intrinsic data types.

A data definition has the following syntax:

[name] directive initializer [,initializer]...

Initializer: At least one initializer is required in a data definition, even if it is zero. Additional initializers, if any, are separated by commas. For integer data types, initializer is an integer constant or expression matching the size of the variable's type, such as BYTE or WORD. If you prefer to leave the variable uninitialized (assigned a random value), the ? symbol can be used as the initializer.

Examples:

value1 **BYTE** 'A'	; character constant
value2 **BYTE** 0	; smallest unsigned byte
value3 **BYTE** 255	; largest unsigned byte
value4 **SBYTE** −128	; smallest signed byte
value5 **SBYTE** +127	; largest signed byte
greeting1 **BYTE** "Good afternoon", 0	; String constant
greeting2 **BYTE** 'Good night'	; String constant

greeting1 **BYTE** 'G','o','o','d'   ; String constant

The hexadecimal codes 0Dh and 0Ah are alternately called CR/LF (carriage-return line-feed) or end-of-line characters.

list BYTE 10,20,30,40   ; Multiple initializers

Note: A question mark (?) initializer leaves the variable uninitialized, implying it will be assigned a value at runtime:

value6 BYTE ?

DUP Operator

The DUP operator allocates storage for multiple data items, using a constant expression as a counter. It is particularly useful when allocating space for a string or array, and can be used with initialized or uninitialized data.

Examples:
v1 BYTE 20 DUP(0)   ; 20 bytes, all equal to zero
v2 BYTE 20 DUP(?)   ; 20 bytes, uninitialized
v3 BYTE 4 DUP("STACK")   ;20 bytes, "STACKSTACKSTACKSTACK"

Operand Types:

As x86 instruction formats:

                [label:] mnemonic [operands][; comment]

Because the number of operands may vary, we can further subdivide the formats to have zero, one, two, or three operands.

Here, we omit the label and comment fields for clarity:

        mnemonic
        mnemonic [destination]
        mnemonic [destination],[source]
        mnemonic [destination],[source-1],[source-2]

x86 assembly language uses different types of instruction operands. The following are the easiest to use:

- Immediate—uses a numeric literal expression
- Register—uses a named register in the CPU
- Memory—references a memory location

Following table lists a simple notation for operands. We will use it from this point on to describe the syntax of individual instructions.

Operand	Description
reg8	8-bit general-purpose register: AH, AL, BH, BL, CH, CL, DH, DL
reg16	16-bit general-purpose register: AX, BX, CX, DX, SI, DI, SP, BP
reg32	32-bit general-purpose register: EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP
reg	Any general-purpose register
sreg	16-bit segment register: CS, DS, SS, ES, FS, GS
imm	8-, 16-, or 32-bit immediate value
imm8	8-bit immediate byte value
imm16	16-bit immediate word value
imm32	32-bit immediate doubleword value
reg/mem8	8-bit operand, which can be an 8-bit general register or memory byte
reg/mem16	16-bit operand, which can be a 16-bit general register or memory word
reg/mem32	32-bit operand, which can be a 32-bit general register or memory doubleword
mem	An 8-, 16-, or 32-bit memory operand

MOV Instruction:

It is used to move data from source operand to destination operand
- Both operands must be the same size.
- Both operands cannot be memory operands.
- CS, EIP, and IP cannot be destination operands.
- An immediate value cannot be moved to a segment register.

Syntax:

        MOV destination, source

Here is a list of the general variants of MOV, excluding segment registers:

```
MOV reg,reg
MOV mem,reg
MOV reg,mem
MOV mem,imm
MOV reg,imm
```

Example:

        MOV bx, 2
        MOV ax, cx

Example:

        'A' has ASCII code 65D (01000001B, 41H)

The following MOV instructions stores it in register BX:

        MOV bx, 65d
        MOV bx, 41h
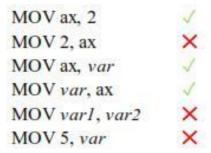   MOV bx, 01000001b
        MOV bx, 'A'

All of the above are equivalent.

Examples:

The following examples demonstrate compatibility between operands used with MOV instruction:

```
MOV ax, 2        ✓
MOV 2, ax        ✗
MOV ax, var      ✓
MOV var, ax      ✓
MOV var1, var2   ✗
MOV 5, var       ✗
```

ADD Instruction

The ADD instruction adds a source operand to a destination operand of the same size. Source is unchanged by the operation, and the sum is stored in the destination operand

Syntax:

ADD dest,source

SUB Instruction

The SUB instruction subtracts a source operand from a destination operand.

Syntax:

SUB dest,source

Sample Program:

```
TITLE Add and Subtract (AddSub.asm)
; This program adds and subtracts 32-bit integers.
INCLUDE Irvine32.inc
.code
main PROC
mov eax,10000h        ; EAX = 10000h
add eax,40000h        ; EAX = 50000h
sub eax,20000h        ; EAX = 30000h

call DumpRegs         ; display registers
exit
main ENDP
END main
```

Lab Exercise:

1. Write an uninitialized data declaration for a16-bit signed integer val1. Initialize 8-bit signed integer val2 with -10.
2. Declare a 32-bit signed integer val3 and initialize it with the smallest possible negative decimal value.

3. Declare an unsigned 16-bit integer variable named wArray that uses three Initializers.

4. Declare a string variable containing the name of your favorite color. Initialize it as a null terminated string.6. Initialize five 16-bit unsigned integers varA, varB, varC, varD & varE with the following values: 12, 2, 13, 8, 14.

5. Convert the following high-level instruction into Assembly
   Language: ebx = { (a+b) – (a-b) + c } +d
   a= 10h , b=15h, c=20h, d=30h

6. Convert the given values of a,b,c,d into binary and then use in 8-bit data definition and implement in the equation.

7. Write a program in assembly language that implements following expression:
   Eax = imm8 + data1 – data3 + imm8 + data2

   Use these data definitions:
   Imm8 = 20
   Data1 word 8
   Data2 word 15
   Data3 word 20