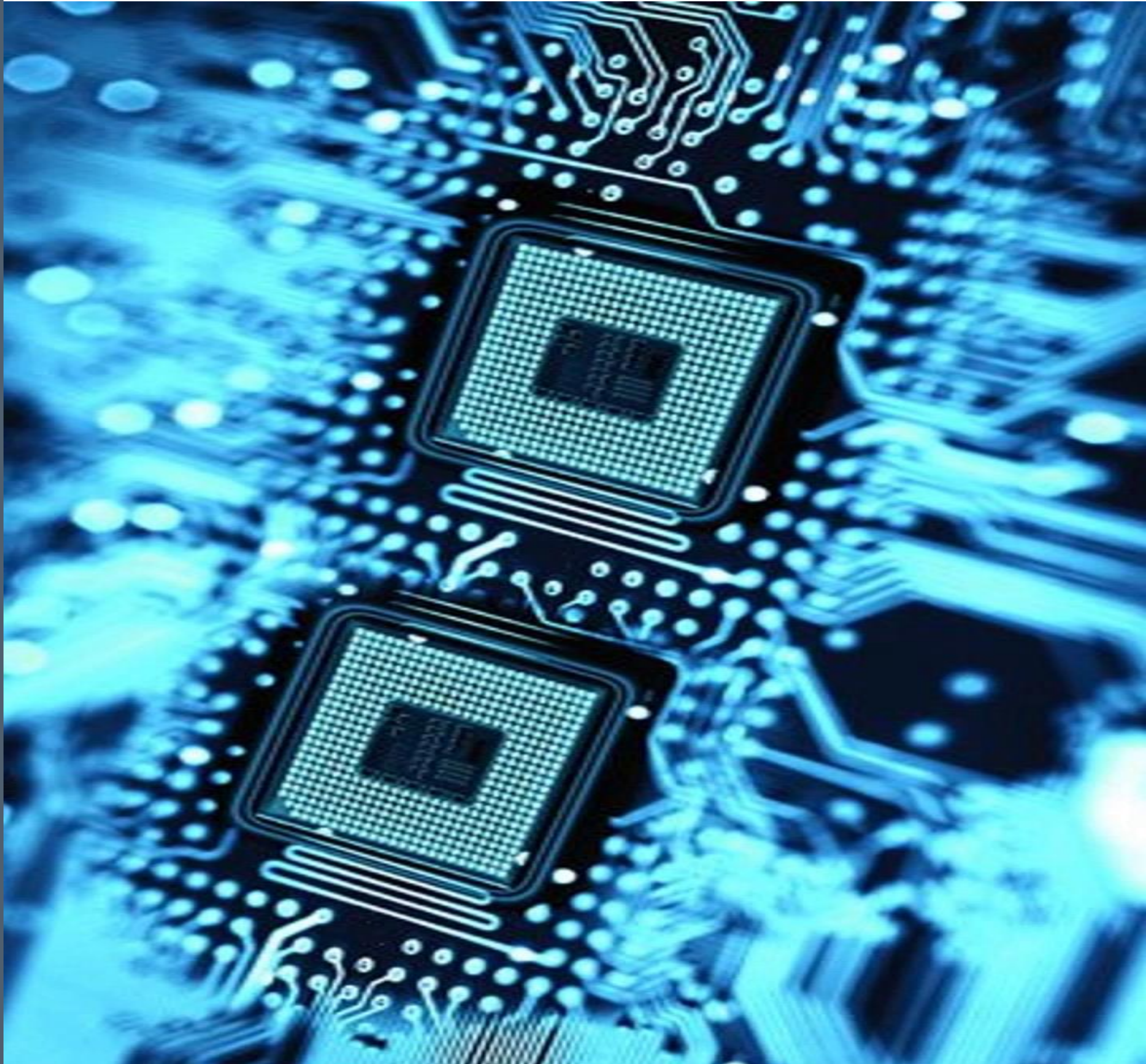
Computer organization & architecture

Course by: Dr. Ahmed Sadek

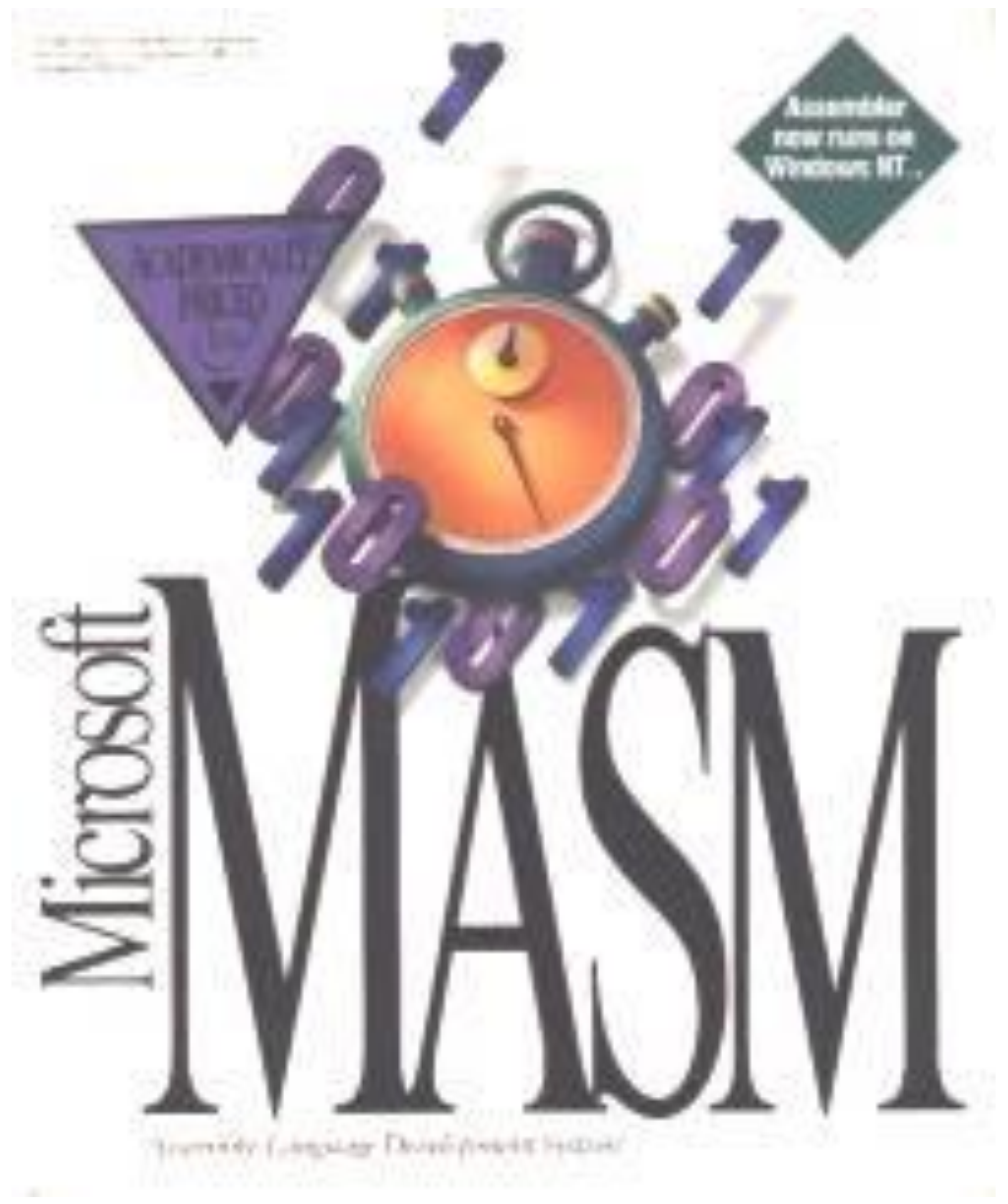
Lab By: Mahmoud Badry

Assembly Language Fundamentals

Chapter 3



About Chapter



- In this chapter, you will learn how to **define** and **declare variables** and **constants**, using Microsoft Assembler **(MASM)** syntax.
- We also can use **Emu8086** but some **difference** occurs.

Program template

```
;Program Description:
;Author:
;Creation date:
;Revisions:
;Date:                  ;Modified by:
.data
;Insert variables here
.code
JMP main
main PROC
;Insert your code here
JMP Exit
main ENDP
;(insert additional procedures here)
Exit: ret
END
```

Defining Data

Chapter 3, Section 3

Intrinsic Data Types

Keyword MASM	Keyword 8086	Usage
BYTE	DB	8-bit
WORD	DW	16-bit
DWORD	DD	32-bit
QWORD	DQ	64-bit
TBYTE	DT	80-bit

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

Data Definition Statement

- A data definition **statement** sets aside **storage** in **memory** for a **variable** and may **optionally** assign a **name** to the variable:

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

- At **least one initializer** is **required** in a data definition, even if it is the ? expression, which does **not assign** a specific **value** to the data.
- All **initializers**, regardless of their number format, are **converted** to **binary** data by the **assembler**.

- **Examples**

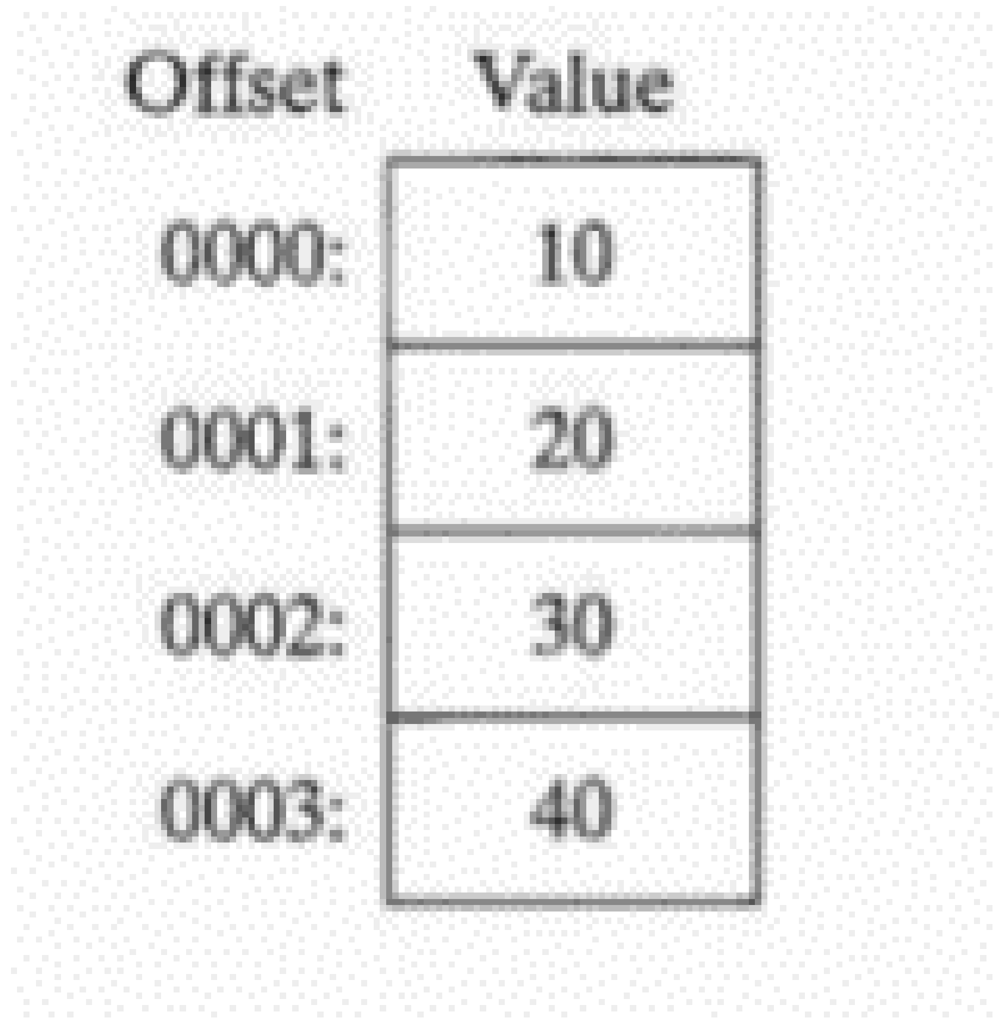
```
value1 DB 'A'      ; character constant

value2 DB 0        ; smallest unsigned byte
value3 DB 255      ; largest unsigned byte

value4 DB ?        ; Empty byte

value5 DB 255      ; unsigned byte
value6 DB -128     ; signed byte
```

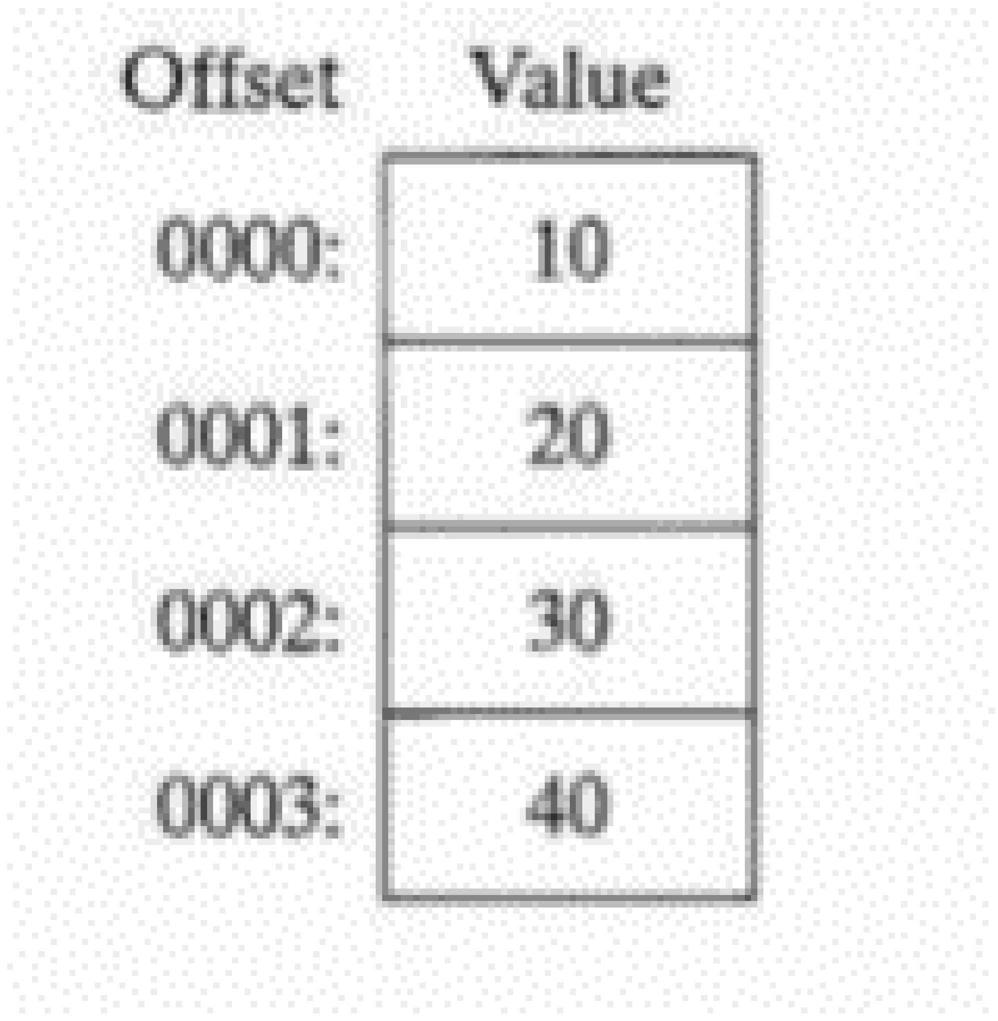
Multiple initializers



- If **multiple** initializers are **used** in the **same** data **definition**, its **label refers** only to the **offset** of the first byte.
- Example:

```
.data
list DB 10,20,30,40
```

Multiple initializers



Offset	Value
0000:	10
0001:	20
0002:	30
0003:	40

- **Not all** data **definitions require labels**. If we wanted to **continue** the **array** of **bytes** begun with list, example:

```
list DB 10, 20, 30, 40
     DB 50, 60, 70, 80
     DB 81, 82, 83, 84
```

- **Within** a **single** data **definition**, its **initializers** can use **different radixes**.

```
list1 DB 10, 32, 41h, 00100010b
list2 DB 0Ah, 20h, 'A', 22h
```

Defining Strings and DUP

- To create a **string** data definition, **enclose** a sequence of **characters** in **quotation** marks. The most common type of **string ends** with a **null** byte, a byte containing the value 0. This type of **string** is **used** by **C/C++,**by **Java**, and by **Microsoft Windows** functions:

```
    greeting1 DB "Good afternoon",0
```

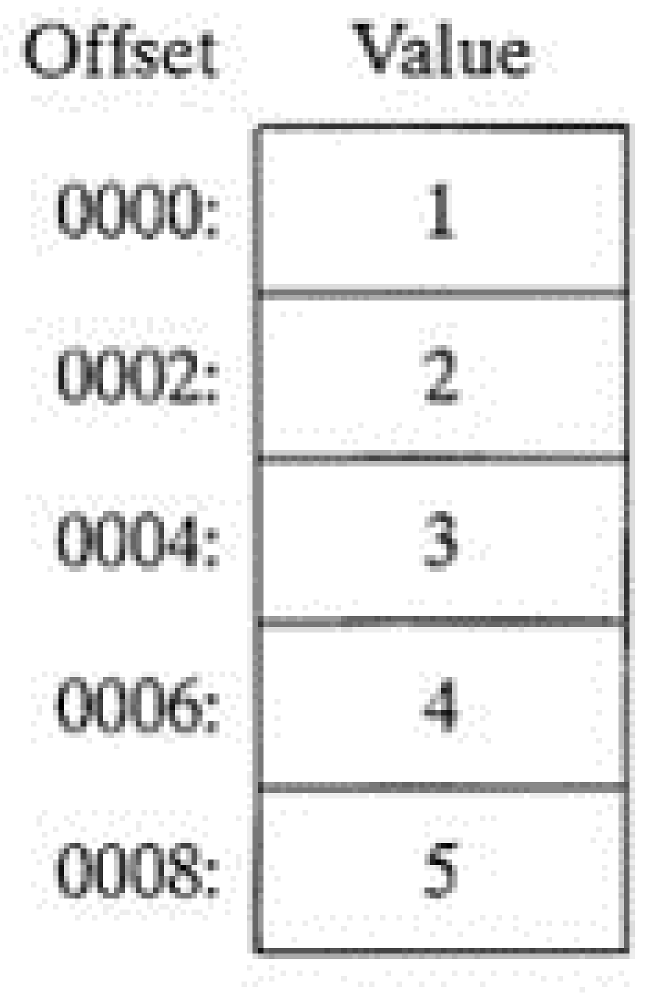
- String multiple lines:

```
greeting2 DB "Welcome to the Encryption Demo program "
          DB "created by Kip Irvine.",0dh,0ah,0
```

- The **DUP** operator generates a **repeated storage allocation**, using a **constant expression** as a counter. It is particularly **useful** when **allocating space** for a string or **array**, and can be used with **Both initialized** and **uninitialized** data **definitions**:

```
 DB 20 DUP(0)            ; 20 bytes, all equal to zero
 DB 20 DUP(?)            ; 20 bytes, uninitialized
 DB 4 DUP("STACK")       ; 20 bytes: "STACKSTACKSTACKSTACK"
```

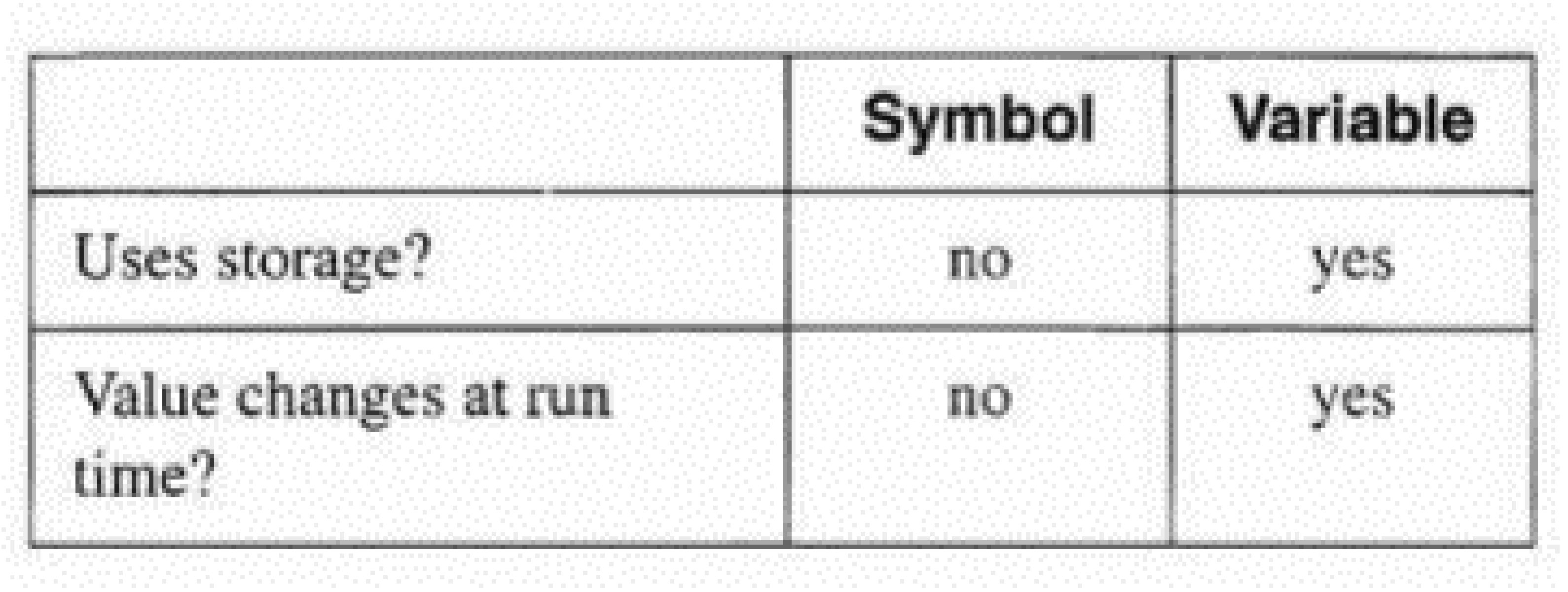
Defining WORD



Offset	Value
0000:	1
0002:	2
0004:	3
0006:	4
0008:	5

```
Val1 DW 65535    ; unsigned
Val2 DW -32768  ;signed
myList DW 1, 2, 3, 4, 5
Array DW 5 DUP( ? )
```

Symbolic Constants



	Symbol	Variable
Uses storage?	no	yes
Value changes at run time?	no	yes

- Created by **associating** an **identifier** (a symbol) with either an **integer expression** or **some text.**
- **Unlike** a **variable** definition, which **reserves storage**, a **symbolic constant** does **not use** any **storage**. Symbols are used only during the assembly of a program, so they **cannot change** at **runtime**.

Equal-Sign Directive

- The **equal-sign** directive associates a **symbol** name with an integer **expression**.

```
name = expression
```

- **Example**:

```
        COUNT = 500
        mov aX,COUNT
```

- We can use the **DUP** operator with the **directive**:

```
        COUNT = 50
        array DW COUNT DUP(0)
```

- **Calculating the Sizes of Arrays and Strings**
- Setting the size **manually**:

```
list DB 10, 20, 30, 40
ListSize = 4
```

- With the **$ operator(current** location counters)

```
list DB 10, 20, 30, 40
ListSize= ($ - list)
```

- To get **size** of **string**

```
myString DB "This string, containing "
         DB "any number  of chars",0
mySting_len = ($ - myString)
```

- **Arrays** of **Words** :

```
list DW 1000h, 2000h, 3000h, 4000h
ListSize = ($ - list) / 2
```

EQU Directive

- The **EQU** directive **associates** a **symbolic name** with either an **integer expression** or some **arbitrary text**. There are three formats:

```
    name EQU expression
    name EQU symbol
    name EQU <text>
```

- EQU can be **useful** when defining any **value** that does **not evaluate** to an **integer**. A **real number** constant:

```
    PI EQU <3.1416>
```

- **Unlike** the **=** directive, a **symbol defined** with **EQU cannot** be **redefined** in the same source code file. This maybe seen as a **restriction**, but it also **prevents** an **existing symbol** from being in advertently **assigned** a **new value**.

- **Example of string usage**

```
pressKey EQU <"Press any key to
continue.",0 >
.
.
.
.data
prompt DB pressKey
```

THANKS